TAMPEREEN TEKNILLINEN YLIOPISTO

JARI SUKSI

# AUDIOLATENSSI MOBIILIALUSTOILLA

Diplomityö

Tarkastaja: professori Tommi Mikkonen
Tarkastaja ja aihe hyväksytty Tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 05.10.2011

# TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
**JARI SUKSI: Audiolatenssi mobiilialustoilla**
Diplomityö, 54 sivua
Huhtikuu 2012
Pääaine: Ohjelmistotuotanto
Tarkastajat: Tommi Mikkonen
Avainsanat: audio, latenssi, mobiili, Qt, Android, iOS, Windows Phone, Symbian, Series 40, BlackBerry, Java ME

Audiokeskeisistä sovelluksista on muodostunut tärkeä segmentti sovelluskaupoille. Toisaalta sovelluskaupat ovat tärkeä osa mobiiliekosysteemejä ja sitä kautta merkittävä vaikuttaja tietyn laitteen tai alustan menestykseen mobiilimarkkinoilla. Kuluttajat ovat jo tottuneet sulavaan käyttötuntumaan ja turhautuvat herkästi, jos vaikutelma ei vastaa totuttua. Esimerkiksi Apple on asettanut riman monessa suhteessa hyvin korkealle eikä vähiten audion toistossa.

Tämän työn keskeisenä teemana on audiolatenssi eri mobiilialustoilla. Yleisesti ottaen latenssi mobiililaitteissa tuntuu olevan suuri etenkin soitannollisesta näkökulmasta. Latenssi koostuu kosketustapahtuman ja äänentoiston aiheuttamasta viiveestä. Lisäksi latenssiin vaikuttaa muu suoritinkuorma. Työssä mitattiin eri mobiilialustojen ja audiokirjastojen soveltuvuutta soitannollisten ohjelmien toteuttamiseen. Tutkittavia alustoja olivat Nokian Series 40-, Symbian-, ja MeeGo 1.2 Harmattan -alustat, Research In Motionin BlackBerry 6.0, Microsoftin Windows Phone 7, Googlen Android sekä Applen iOS.

Latensseja määritettiin yksinkertaisen mittausjärjestelyn avulla, jossa kosketusnäyttöä napautetaan voimakkaasti, minkä johdosta mittauksia varten alustalle toteutettu audiosovellus toistaa lyhyen äänimerkin. Tapahtumaketju nauhoitetaan mikrofonilla ja nauhoitteen aaltokuviosta mitataan napautuksen ja äänimerkin aikaväli. Mittauksia varten valittiin suorituskyvyltään mahdollisimman tasavertaisia laitteita kahteen eri testiryhmään, joista toinen edustaa mallistojen heikkotehoisempia laitteita ja toinen tehokkaimpia laitteita.

Mitatut latenssit ja keskihajonnat on koottu taulukkoon, josta voidaan löytää sopivimmat konfiguraatiot alhaisen latenssin saavuttamiseksi. Tulosten perusteella on analysoitu alustakohtaisesti eri kirjastojen soveltuvuutta alhaista audiolatenssia vaativiin ohjelmistoihin. Lisäksi kirjastojen ja rajapintojen käyttöä on havainnollistettu koodiesimerkein.

# ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
**JARI SUKSI : Audio latency on mobile platforms**
Master of Science Thesis, 54 pages
April 2012
Major: Software engineering
Examiner: Tommi Mikkonen
Keywords: audio, latency, mobile, Qt, Android, iOS, Windows Phone, Symbian, Series 40, Java ME

Audio focused applications have become an important segment for mobile app stores. In addition, app stores are a crucial part of mobile ecosystems and therefore play a big part in whether a device or platform succeeds on the mobile market. Users have already got used to fluid user experience and will easily get frustrated, if the impression doesn't correspond to the expected. For example Apple has set the bar high and not least over audio playback.

This thesis is about audio latency on mobile platforms. Generally speaking, the latency seems to be quite disruptive at least for musical applications, that require keeping up to a certain rhythm for example. The overall latency consists of touch latency and the delay caused by audio playback along with other processor load. This study evaluates the feasibility of various mobile platforms and audio libraries for implementing instrumental applications. The platforms under investigation were Nokia Series 40, Symbian and MeeGo 1.2 Harmattan, BlackBerry 6.0 by Research In Motion, Microsoft Windows Phone 7, Google Android and Apple iOS.

Latencies were measured using a simple experimental arrangement, where a touch screen gets tapped strongly and as a result an audio application, implemented just for the measuring purposes, plays a sound effect. The chain of events is being recorded with a microphone. Then the time difference between tap impact and sound effect can be measured from the wave form using an audio editor such as Audacity. Two groups of devices were selected for the measurements: one that represents the flagship smartphones of the vendors and one with lower priced devices, that generally do not perform as well as the more pricy models.

The measured latencies and standard deviations have been gathered into a table, which can be examined for the best configurations in aim of gaining low latency. Additionally, the audio libraries have been compared to each other in light of the measured latencies and the usage of the libraries has been demonstrated with short code examples.

# ALKUSANAT

Tämä työ on tehty Futurice Oy:lle osana uutta diplomityön ohjausmallia. Kahden kuukauden intensiivijakson aikana sain työni hyvään vauhtiin, vaikka työ ei aivan valmistunutkaan tavoiteajassa. Aihevalinta juontui Nokia Oyj:lle syksyn 2010 ja kesän 2011 aikana tuotetuista Drumkit-esimerkkisovelluksista Series 40-, Symbian-, Harmattan- ja Windows Phone -alustoille.

Työtä on ohjannut Tampereen Teknillisen Yliopiston professori Tommi Mikkonen sekä Futurice Oy:n puolesta tekniikan tohtori Risto Sarvas. Haluan kiittää molempia hyvistä neuvoista ja kokeneesta ohjauksesta. Lisäksi haluan kiittää Riku Valtasolaa, joka auttoi minua diplomityöaiheen löytämisessä, sekä Kari-Pekka Koljosta avusta Qt-toteutusten tekemisessä. Kiitos myös loistaville työkavereilleni, jotka uhrasivat aikaansa työni tarkastamiseen. Joukkoistamisessa on voimaa!

Kiitos kuuluu myös vanhemmilleni ja sisaruksilleni, jotka ovat auttaneet minua asettamaan tavoitteita opiskelun suhteen ja tukeneet minua kaikin tavoin. Tämä diplomityö on omistettu aina hyväntuulisille veljentyttärilleni Tarulle ja Sannalle. Vaikka kirjasta tuli vähän tylsä, toivottavasti se edes jollain tasolla innostaisi teitä opiskelemaan ahkerasti heti ekasta luokasta lähtien – voin kertoa, että kannattaa.

Tampereella, 13.3.2012

Jari Suksi

# SISÄLLYS

# 1. JOHDANTO

Audiokeskeisistä sovelluksista on muodostunut tärkeä segmentti sovelluskaupoille. Toisaalta sovelluskaupat ovat kriittinen osa mobiiliekosysteemejä ja sitä kautta merkittävä vaikuttaja tietyn laitteen tai alustan menestykseen. Mobiilimarkkinoilla kuluttajat ovat jo tottuneet sulavaan käyttötuntumaan ja turhautuvat herkästi, jos vaikutelma ei vastaa totuttua. Esimerkiksi Apple on asettanut riman monessa suhteessa hyvin korkealle, eikä vähiten audion toistossa.

Mobiilimarkkinoilla kilpailu on kovaa ja uusia laitemalleja ilmestyy kuin sieniä sateella. Etenkin avoimet käyttöjärjestelmät ovat madaltaneet laitevalmistajien kynnystä tuoda omia laitteitaan myyntiin. Pelkästään hyvä laitteisto ja hyvä käyttöjärjestelmä ei enää välttämättä kuitenkaan riitä mobiilimarkkinoilla, vaan tärkeässä roolissa ovat myös eri alustoille suunnatut sovelluskaupat ja niiden sovellusvalikoimat. Osa valikoimaa ovatkin erilaiset soitto-ohjelmat ja syntetisaattorisovellukset, jotka vaikuttavat olevan melko oleellinenkin osa ohjelmatarjontaa. Esimerkiksi Smulen Ocarina-sovellus kuuluu Applen App Storen kaikkien aikojen Top 20 -listalle.

Tämän työn keskeisenä teemana on audiolatenssi eri mobiilialustoilla. Latenssi on aika, joka kuluu tapahtuman ja sen vaikutusten välillä. MIDI-maailmassa latenssit ovat olleet jo pitkään keskustelunaiheena. Kun pianisti painaa MIDI-kontrollerin kosketinta, vaaditaan tietty määrä prosessointia ennenkuin painallusta vastaava ääni on kuultavissa kaiuttimista. Tämä viive on hyvin kriittistä soittotuntuman kannalta. Mobiilisovelluksissa laitteen näytön kosketus vastaa MIDI-kontrollerin painallusta ja se miten kauan äänen toistamisen aloittaminen kestää on oleellista koko sovelluksen käyttötuntuman kannalta. Liian iso latenssi tekee sovelluksesta suorastaan käyttökelvottoman.

Yleisesti ottaen latenssi mobiililaittessa tuntuu olevan suuri etenkin soitannollisesta näkökulmasta. Ammattimuusikosta jo 10 ms latenssi voi alkaa tuntua häiritsevältä, kun taas mobiililaitteella alle 10 ms viiveisiin on hankala päästä missään olosuhteissa, eikä se oikeastaan ole tarpeenkaan. Kuitenkin monella alustalla latenssi on kohtuuttomankin suuri. Viive syntyy kosketustapahtuman viiveestä ja äänentoiston viiveestä, sekä muun prosessorikuorman aiheuttamasta suoritusviiveestä.

Työssä selvitettiin mille mobiilialustoille on tällä hetkellä mahdollista toteuttaa alhaista audiolatenssia vaativa sovellus, ja mitkä seikat yleisesti vaikuttavat latenssiin. Samalla ajatuksena oli kartoittaa alustoille tarjolla olevia kirjastoja ja koittaa

löytää selkeitä suuntaviivoja latenssikriittisen sovelluksen toteuttamiseksi eri alustoille. Työn edetessä huomio kiinnittyi lisäksi latenssien hajontaan, jolla on itse latenssin ohella suuri vaikutus audiokeskeisten sovellusten käytettävyyteen.

Työn motivaattorina ovat olleet syksyn 2010 ja kesän 2011 aikana toteutetut Drumkit-ohjelmat. Drumkitissä soitetaan rumpuääniä mustia rumpualustoja painelemalla. Ensimmäinen versio Drumkitistä toteutettiin Java ME:llä Series 40 -alustalle. Toteutus demonstroi muun muassa Mobile Media API 1.2:n käyttöä sekä kohdelaitteena olleen Nokia X3 Touch and Type -puhelimen kosketusnäytön hyödyntämistä. Drumkit on toteutettu myös Windows Phone 7 -alustalle sekä Qt-versiona Nokian Symbian- ja MeeGo-alustoille. Tärkeänä osana sovellusten kehitystä on ollut pyrkiä pitämään äänentoistolliset viiveet mahdollisimman pieninä, mutta nopeasti kävi ilmi, että audiolatenssit vaihtelevat alustoittain melkoisesti, eikä tämän tyyppisen sovelluksen toteuttaminen välttämättä ole edes mahdolista kaikille alustoille.

Tutkimusmenetelmä on verrattain suoraviivainen. Kokonaislatenssia on mitattu siten, että kosketusnäyttöä napautetaan voimaakkaasti, minkä johdosta toistetaan terävä ääninäyte. Tapahtumaketju on nauhoitettu mikrofonilla ja mitattu latenssi suoraan nauhoitteen aaltokuviosta käyttäen Audacity-äänieditoria. Mittauksia varten kaikille tarkasteltaville alustoille on toteutettu minimalistinen äänisovellus, jossa on näyttöä koskettamalla toistetaan lyhyt ääni-impulssi. Tutkimuksen piiriin kuuluvia alustoja ovat Series 40, Symbian[1], Symbian[3], Harmattan 1.2 MeeGo, Windows Phone 7 ja 7.5, Android 2.3, iOS 5 sekä BlackBerry 6.0. Vastaavasti on mitattu myös Drumkitin latensseja, että on voitu todeta mahdolliset kehitystarpeet. Vertailun vuoksi latensseja on mitattu myös olemassa olevilla Drumkitin tyyppisillä sovelluksilla, joita esimerkiksi Google Play -sovelluskaupasta (entinen Android Market) löytyy useampia.

Mittauksia varten on valittu suorituskyvyltään mahdollisimman tasavertaisia laitteita kahteen eri testiryhmään, joista toinen edustaa mallistojen heikkotehoisempia laitteita ja toinen tehokkaimpia laitteita. Näin voidaan todeta myös laitteiston vaikutus latenssiin. Audioformaattina on käytetty hyvin pääasiassa Wave-formaattia, koska valitut laitteet ja kirjastot tukevat sitä poikkeuksetta. Tosin suoratoistorajapintoja varten audio data täytyy toisinaan itse lukea tiedostosta.

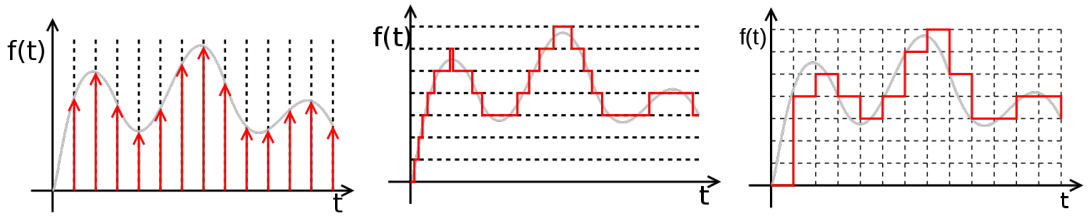
Luvussa 2 tutustutaan digitaaliseen audioon ja siihen liittyviin ohjelmoinnin erityispiirteisiin. Luvussa 3 tutustutaan Drumkit-sovelluksiin, sekä kahteen muuhun latenssikriittiseen muusiikkisovellukseen. Seuraavaksi luvussa 4 käydään läpi tutkittavat mobiilialustat, minkä jälkeen tarkastellaan alustoille saatavilla olevia audiokirjastoja luvussa 5. Luku 6 määrittelee latenssimittauksiin käytetyt tutkimusmenetelmät. Lopulta luvussa 7 on esitelty ja analysoitu työn tulokset. Mitatut latenssit on kerätty taulukkoon, jonka pohjalta on arvioitu kirjastojen soveltuvuutta alhaista latenssia vaatiiviin sovelluksiin. Luku 8 sisältää vielä lyhyen yhteenvedon.

# 2. DIGITAALINEN AUDIO JA OHJELMOINTI

Äänen varastointi ja käsittely on helpottunut digitalisaation myötä huomattavasti. Bittimuodossa olevaa audiota voidaan säilyttää ja kopioida muuttumattomana periaatteessa ikuisuuden siinä missä esimerkiksi vanha C-kasettinauhoite on selvästi heikentynyt ajan saatossa ja jokaisella kopiointikerralla laatu kärsinyt. Audio on pitkään ollut eräs tärkeä signaalinkäsittelyn tutkintakohde. Kehitys onkin mahdollistanut esimerkiksi digitaalisten efektien käytön muun muassa live-musiikissa. Reaaliaikaisissa sovelluksissa digitaalinen äänenkäsittely aiheuttaa kuitenkin aina latenssia. Seuraavissa kohdissa tutustutaan lyhyesti äänisignaalin digitoimiseen, audion latenssiin, sekä audio-ohjelmointiin.

## 2.1 Digitaalinen audio

Ääni koostuu ilmanpaineen vaihteluista ja voidaan muuntaa elektroniseen muotoon esimerkiksi mikrofonin avulla. Tällöin ääni muunnetaan jännitteenvaihteluiksi. Ennen muunnosta ääni on analogisessa muodossa. Kun jatkuvaa analogista äänisignaalia halutaan käsitellä tietokoneella, tulee signaali näytteistää digitaaliseen eli diskreettiin esitysmuotoon. Tätä kutsutaan analogi-digitaalimuunnokseksi (ADC). Kuva 2.1 havainnollistaa AD-muunnosprosessia. [36]




**Kuva 2.1:** Analogisen signaalin näytteistäminen ja kvantisointi tuottavat signaalin, joka voidaan esittää digitaalisesti

Analogisignaalista otetaan tasaisin väliajoin näytteitä, jotka kvantisoidaan sopivaan tarkkuteen siten, että näytteen magnitudi voidaan esittää halutulla määrällä bittejä. [36] Jos näyte esitetään esimerkiksi yhdellä tavulla eli käyttäen kahdeksaa bittiä, voi näyte saada etumerkillisessä esitysmuodossa arvon väliltä -127–128 tai etumerkittömässä esityksessä arvon väliltä 0–255. AD-muunnoksen jälkeen signaali on niin sanotussa pulssikoodimoduloidussa (PCM) muodossa.

Kvantisointi aiheuttaa äänisignaaliin jonkin verran kohinaa. Mitä enemmän bittejä on käytössä näytteen tallentamiseen, sitä pienempi on kvantisoinnin aiheuttama virhe, ja sitä kautta myös kohinan suhteellinen osuus on pienempi. Näytteenottotaajuus puolestaan sanelee kuinka korkeita äänentaajuksia signaalista voidaan tallentaa. Nyquistin teoreeman mukaan näytteenottotaajuuden tulee olla vähintään kaksi kertaa niin suuri kuin signaalin korkein taajuus, jotta vältytään signaalin laskostumiselta ja varmistutaan näin, että näytteiden perusteella voidaan vielä tuottaa alkuperäisen kaltainen signaali. Laskostumiselta vältytään, kun signaalista alipäästösuodatetaan liian korkeat taajuudet pois. [36]

Audiosignaali voi sisältää useita kanavia. Järjestelmiä, joissa kanavia on enemmän kuin yksi kutsutaan yleisesti monikanavajärjestelmiksi. Käytettäessä useampia kanavia ääni voidaan lähettää kuulijalle eri suunnista. Musiikissa on jo pitkään käytetty kahta kanavaa. Tällöin puhutaan stereoäänestä. Elokuvissa on käytetty monesti neljää tai useampaa kanavaa. Yksikanavainen eli monoauraalinen ääni voi kuulostaa keinotekoiselta ja värittömältä. [36] Mobiililaitteissa usemmasta kanavasta on hyötyä vain käytettäessä kuulokkeita tai kytkettäessä laite ulkoiseen audiojärjestelmään. Niinpä yhden kanavan käyttö on monesti hyvä tapa minimoida audiotiedostojen kokoa.

Kun audio on digitaalisessa muodossa, sitä voidaan käsitellä ohjelmallisesti. Taajuuskorjaus ja erilaiset suotimet ovat hyvin usein käytettyjä toimenpiteitä. Lisäksi voidaan säädellä signaalin sävelkorkeutta, vahvistaa signaalia, lisätä kaikua ja niin edelleen. Signaalin pohjalta voidaan myös tuottaa tajuusjakaumia ja tehdä muita analysoivia toimenpiteitä.

Digitaalisen signaalin toistamiseen sisältyy myös useita vaiheita. Viimeisessä vaiheessa digitaalinen esitys muunnetaan takaisin analogiseksi ja ohjataan kaiuttimelle tai muuhun ulostuloon. Muunnosta kutsutaan digitaali-analogimuunnokseksi (DAC). Jos toistettavia signaaleja on useita, ne täytyy ensiksi yhdistää eli miksata yhdeksi signaaliksi. Miksausta voidaan suorittaa jo omassa koodissa, mutta jos se ei ole välttämätöntä on miksaaminen parempi jättää mahdollisimman lähelle laitteistoa eli mielellään ajurin huolehdittavaksi, jolloin voidaan mahdollisesti hyötyä myös laitteiston tarjoamista signaalinkäsittelyominaisuuksista. Valitettava trendi on, että heikosti toimivia ajureita koitetaan paikata lisäämällä uusia kirjastoja ja kerroksia audioarkkitehtuuriin. Esimerkiksi Linuxille on ilmestynyt useita kirjastoja kuten OSS, ALSA, Jack, Pulse Audio, ESD, ART sekä GStreamer, vaikka hyvä riittäisi. [35]

Jotta kaiuttimesta kantautuva ääni kuulostaisi yhtenäiseltä, on audiolaitteelle taattava katkeamaton bittivirta. Useimmiten ohjelman halutaan tekevän rinnakkaisia tehtäviä, eikä keskittyvän pelkästään audion toistoon. Tällöin ei voida taata, että prosessori ehtisi aina palvelemaan audion toiston tarpeita oikea-aikaisesti. Tästä

syystä signaalia luetaan puskuriin sopivan kokoisissa paloissa. Puskuroinnilla (buffering) pidetään huoli, että laitteelle on jatkuvasti dataa tarjolla, eikä ääni pääse katkeilemaan.

Dan Connor vertaa blogimerkinnässään audiopuskuria ämpäriin, johon virtaa vettä. Ämpärin pohjassa on reikä, josta vesi virtaa eteenpäin. Vaikka virta ämpäriin katkeaa, ämpärin pohjassa olevasta reiästä vesi virtaa vielä jonkin aikaa. Niinpä jos ämpäriin alkaa taas virrata uutta vettä ennen kuin ämpäri ehtii tyhjentyä kokonaan, säilyy pohjan reiästä tuleva virta katkeamattomana. [7]

## 2.2 Audiolatenssi

Audiolatenssia syntyy aina kun audiosignaalia käsitellään. Jos esimerkiksi kierrätetään ääni tietokoneen läpi siten, että nauhoitetaan ääni mikrofonilla ja toistetaan se suoraan tietokoneen kaiuttimista muodostuu latenssia ainakin seuraavissa vaiheissa: AD-muunnoksessa, sisäänmenopuskurissa, ääniajurissa, ulostulopuskurissa sekä DA-muunnoksessa. Jos signaalia vielä muokataan jollain tavalla, kasvaa latenssi entisestään. AD- ja DA-muunnosten latenssit ovat hyvin pieniä – tyypillisesti noin 1–2 ms. [2]

Oleellisin kohta latenssin kannalta on audiopuskuri, sillä sen koko vaikuttaa suoraan audion latenssiin. Palataanpa vielä Connorin vesiämpärivertaukseen. Ajatellaan, että ämpäriin virtaavaan veteen kaadettaisiin väriainetta. Vierähtää tovi, ennen kuin väriaine alkaa näkyä pohjan reiästä tulevassa vedessä. Mitä isompi ämpäri on, sitä kauemmin kestää, ennen kuin vaikutukset näkyvät ulostulevassa virrassa. [7]

Puskurin koko on siis suoraan verrannollinen latenssin kokoon. Puskuria ei kuitenkaan voida pienentää mielivaltaisen pieneksi, sillä prosessorin kuormasta riippuen puskuri saattaa ehtyä (underrun) ennen kuin uutta dataa on saatavilla. Pieni latenssikaan ei siis automaattisesti ole parempi, vaan puskurin koko on syytä valita sovellusalueen mukaan. Esimerkiksi taustamusiikin soittamisessa ei latenssi ole kovin tärkeässä asemassa. Tällöin voidaan valita hyvin isompikin puskurikoko. Samoin ääntä nauhoitettaessa latenssia tärkeämpää on, että nauhoite säilyy ehyenä. Myös tällöin on syytä käyttää tarpeeksi isoa puskuria.

Aina puskurin kokoon ei ole mahdollista vaikuttaa, ja vaikka olisikin, voi pienin mahdollinen puskurin koko olla edelleen liian iso sovelluksiin, joissa alhainen latenssi on avaintekijä. Tällöin latenssia voidaan koittaa pienentää myös nostamalla näytteenottotaajuutta. Havainnollistetaanpa tilannetta jälleen vesiämpärivertauksella. Ämpärin pohjassa olevaa reikää isontamalla vesi alkaa virrata nopeammin. Niinpä ämpäri myös tyhjenee nopeammin ja sitä kautta veden värjäytyminen näkyy ulostulevassa virrassa kestää vähemmän aikaa. Puskuroinnin aiheuttama latenssi saadaankin jakamalla puskurin koko näytteenottotaajuudella. CD-levyllä käytettävä PCM-signaali on näytteenottotaajuudeltaan 44,1 kHz ja jokainen näyte esitetään 16 bitillä

per kanava [36]. Jos puskuri on asetettu esimerkiksi 512 näytteen kokoiseksi, 44,1 kilohertsin näytteenottotaajuudella latenssiksi muodostuu noin 12 ms. Nostamalla näytteenottotaajuus 88,2 kilohertsiin, latenssi puolittuu noin 6 millisekuntiin.

Nolla-latenssiin on mahdoton päästä, eikä se ole tarpeenkaan. Äänen nopeus ilmassa on noin 340 m/s, joten jo 34 senttimetrin matkalla muodostuu 1 ms viive ja 3,4 metrin matkalla viive kasvaa jo 10 millisekuntiin. Esimerkiksi konserttisalissa soittaja saattavat olla hyvinkin etäällä toisistaan, jolloin viiveet voivat kasvaa verrattain suuriksi. Miten he sitten pysyvät tahdissa? Tietysti kapellimestari ohjaa soittoa, mutta viiveet vaikuttavat yhtä lailla kappelimestariin. Stanfordin yliopiston musiikin laitoksen tekemän tutkimuksen mukaan soittajilla ei ole ongelmia pysyä tahdissa, vaikka viiveet olisivat jopa 40 millisekunnin luokkaa tai enemmän. Itse asiassa tutkimus osoittaa, että 10–20 ms viivellä on stabiloiva vaikutus tempoon. Lievä latenssi on siis jopa suotavaa. [16]

Ihminen osaa alitajuntaisesti jossain määrin adaptoitua latenssiin ja toisaalta esimerkiksi ajoittaa motoriset liikkeet ennenaikasesti siten, että soitto osuu rytmiin. Ongelmallisempaa on kuitenkin latenssin hajonta (jitter), joka voi soittotuntuman kannalta olla jopa häiritsevämpää kuin latenssi itsessään. [18] Jos latenssi pysyy vakiona, voidaan monessa sovelluksessa eliminoida latenssin vaikutusta ajoituksen enneaikaistamisella. Hajonta kuitenkin hankaloittaa tilannetta, sillä latenssin vaihdellessa, ei voida tarkkaan ennustaa paljonko ajoitusta tulisi korjata.

Aivot hyödyntävät viivettä äänilähteen sijainnin määrittämiseen muiden niin sanottujen binauraalisten vihjeiden ohella. Tätä kutsutaan Haas-efektiksi. [16] Ääni saapuu korviin eriaikaisesti, jos kuulijan pää on kulmassa äänilähteeseen nähden. Aivot pystyvätkin havaitsemaan jopa 20 nanosekunnin poikkeamia. Niinpä lievät laitteiston epätarkkuudesta johtuvat näytteenottotaajuuden muutoksetkin saattavat aiheuttaa ongelmia sijainnin määrittämisessä. [18] Näin hienojakoisella virheellä ei kuitenkaan ole juuri vaikutusta järjestelmän responsiivisuuteen.

## 2.3 Audio-ohjelmointi

Mobiilialustat tarjoavat useita tapoja toistaa audiota. Äänilähteenä voidaan käyttää laitteeseen asennettuja tai internetistä löytyviä äänileikkeitä eri formaateissa, niin pakkaamattomassa (wav) kuin pakattussakin muodossa (mp3). Ääntä voidaan toisaalta myös generoida ohjelmallisesti esimerkiksi luomalla tietyn taajuinen sini-aalto ja näytteistämällä se tavuiksi. Moni alusta tarjoaakin rajapinnan, jonka kautta voidaan soittaa perusnuotteja. Toisinaan toistettavaan audioon on myös mahdollista lisätä erilaisia efektejä kuten kaikua. Monesti äänikirjastot tukevat myös midi-toistoa. Midejä voidaan ladata kokonaisina leikkeinä ja usein rajapinnan avulla on mahdollista soittaa myös yksittäisä nuotteja lähettämällä niin sanottuja midi-viestejä.

Yksinkertaisimmillaan äänileikkeen toistaminen tapahtuu siten, että luodaan soi-

tinluokan (player) instanssi, jolle annetaan äänileikkeen polku. Tämän jälkeen leike voidaan toistaa kutsumalla soittimen play-metodia. Usein rajapinta tarjoaa myös prefetch-metodin, jolla pyydetään soitinta noutamaan äänileike muistiin etukäteen. Näin toisto voidaan aloittaa nopeasti.

Suoratoisto (streaming) on menetelmä, jossa äänileikkeen data luetaan puskuriin pienissä paloissa ja lähetetään äänilaitteiston toistettavaksi. Käytännössä tämä tapahtuu rajapinnalta tulevan takaisinkutsun avulla. Tekniikka on mielenkiintoinen nimenomaan alhaisen latenssin kannalta, sillä käyttämällä pientä puskuria, voidaan audion toisto aloittaa hyvin nopeasti lukematta koko leikettä kerralla muistiin. Tekniikka on hyvin laajalti käytetty myös toistettaessa audiota tai videota verkon yli. Tällöin käytetty purkurin koko on verkon viiveistä johtuen monesti huomattavan iso.

# 3. LATENSSIKRIITTISIÄ MOBIILISOVELLUKSIA

Latenssikriittisiksi sovelluksiksi voidaan lukea kaikki interaktiiviset audiosovellukset kuten virtuaaliset syntetisaattorit tai vastaavat soittimelliset sovellukset, joilla voidaan toistaa ääniä käyttäjän syötteen mukaan reaaliajassa. Käytännössähän reaaliaikaisuus on vain tavoite, johon on mahdotonta päästä, sillä digitaalisessa prosessoinnissa viiveitä syntyy aina. Mobiililaitteille on tehty paljon erilaisia esimerkiksi pianoa, rumpuja tai vaikkapa okarinaa mallintavia soitto-ohjelmia, joissa kaikissa on tärkeää soiton oikea-aikaisuus ja mahdollisuus rytmin ylläpitämiseen. Tässä luvussa tutustutaan Drumkit-esimerkkisovelluksiin, menestyksekkääseen Ocarina-sovellukseen sekä Loopy-moniraituriohjelmaan. Luvussa 7 on myös mitattu latensseja joukolle eri sovelluskaupoista saatavia enemmän tai vähemmän latenssikriittisiä mobiilisovelluksia.

## 3.1 Drumkit

Drumkit on ohjelma, jolla käyttäjä voi soittaa rumpuääniä kosketusnäytön avulla. Ensimmäinen versio Drumkitistä toteutettiin Java ME:llä Series 40 -alustalle. Toteutus demonstroi muun muassa Mobile Media API:n käyttöä sekä kohdelaitteena olleen Nokia X3 Touch and Type -puhelimen kosketusnäytön hyödyntämistä.

**Kuva 3.1:** Drumkitin Windows Phone 7 -versio

Drumkitissä soitetaan rumpuääniä mustia rumpualustoja painelemalla. Alustoihin voi vaihtaa eri perkussioääniä ja käyttäjä voi myös nauhoittaa soittoaan. Lisäksi nauhoitteen päälle voi soittaa. Drumkit on toteutettu myös Windows Phone 7

-alustalle sekä Qt-versiona Nokian Symbian- ja Harmattan-alustoille. Uudempiin Drumkit-versioihin on toteutettu myös niin sanottu rumpusetti-näkymä, jossa kaikki perkussio-äänet ovat valmiiksi sormien ulottuvilla.

Tärkeänä osana sovelluksen kehitystä oli pyrkiä pitämään äänentoistolliset viiveet pieninä, mistä sitten muodostuikin tämän diplomityön motivaattori. Yksittäisiä ääniä soitettaessa latenssi ei niinkään häiritse, mutta yritettäessä soittaa esimerkiksi rumpukomppia, tulee viiveen vaikutus välittömästi esiin. Tempon pitäminen tasaisena muuttuu vaikeaksi, koska itse komppi laahaa jäljessä. Mitä nopeammin yrittää soittaa, sitä vaikeampi rytmiä on ylläpitää. Hyvin hitaasti soitettaessa viiveen häiritsevä vaikutus lievenee.

## 3.2 Ocarina

Ocarina on eräs Applen sovelluskaupan myydyimmistä sovelluksista. Se tuottaa säveliä muuttamalla mikrofonin tallentaman puhallusäänen okarina-huilua imitoivaksi ääneksi. Sävelkorkeutta vaihdellaan kosketusnäytöllä olevia “reikiä” peittämällä samaan tapaan kuin oikeallakin okarinalla. Audiolatenssin kannalta Ocarina on hyvin kriittinen, sillä puhallusäänen analysointi ja prosessointi aiheuttaa lisäviivettä. [43]

**Kuva 3.2:** SMulen Ocarinan käyttöliittymä

Ocarinan on tehnyt SMule-niminen ohjelmistotalo, joka on erikoistunut pääasiassa iPhonelle ja iPadille suunnattuihin interaktiivisiin audio- ja videosovelluksiin. SMulen muita tunnettuja sovelluksia ovat esimerkiksi Leaf Trombone, joka on hyvin läheistä sukua Ocarinalle, Magic Piano sekä auto-tune sovellus I Am T-Pain. Lisäksi yritys on tuonut markkinoille iPadille tarkoitetun sovelluksen, jolla voi luoda musiikkia lyhyitä videonäytteitä toistamalla. [37]

## 3.3 Loopy$^2$

Älypuhelimiin on saatavilla useita erilaisia niin sanottuja "luuppaus"-sovelluksia, joiden ajatuksena on yhdistellä usein melko lyhyitä musikaalisia silmukoita siten, että syntyy yhtenäinen musiikkiteos. Yksittäinen silmukka voidaan tuottaa ohjelmallisesti määrittelemällä soitettavien ääninäytteiden, kuten esimerkiksi rummuniskujen, ajankohdat. Toinen vaihtoehto on nauhoittaa ääntä mikrofonin kautta.

Loopy$^2$ on esimerkki tällaisesta sovelluksesta iPhonelle. Sen avulla käyttäjä voi nauhoittaa soittoaan, lauluaan tai vaikkapa niin sanottua beatboxausta. Yhtäaikaa soivia raitoja voi olla korkeintaan kuusi kappaletta. Raitoja voi kuitenkin yhdistää toisiinsa, joten käytännössä nauhotteita voi olla useampiakin. [1]

**Kuva 3.3:** Everyday Looperin ja Loopy$^2$:n käyttöliittymät

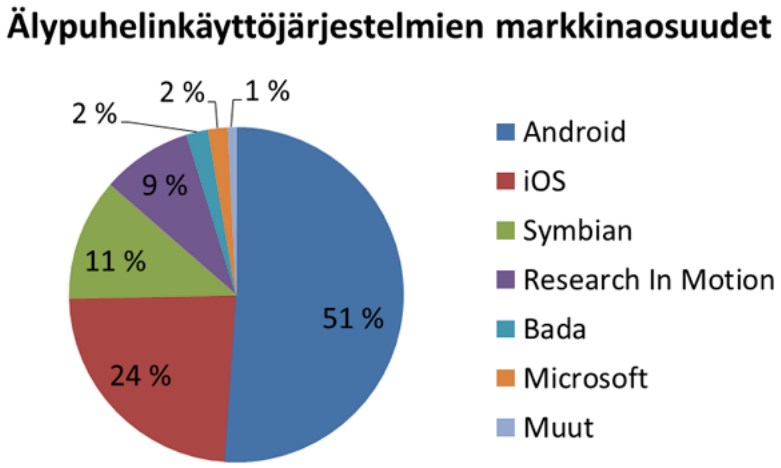
Loopyn käyttöliittymä poikkeaa hieman perinteisemmistä sekvensseriohjelmista. Loopyssa raidat kuvataan ympyröinä, kun sen sijaan esimerkiksi Everyday Looper -sovelluksessa nauhoitetut raidat kuvataan horisontaalisina palkkeina [9] kuten kuvasta 3.3. Käytännön kannalta ero ei kuitenkaan ole kovinkaan suuri, vaan kyse on pikemminkin visuaalisesta vetoavuudesta.

Myös edellä kuvatun kaltaisessa sovelluksessa latenssi on tärkeässä roolissa. Kuitenkin lantessin vaikutus voidaan jossain määrin eliminoida silmukoiden synkronisoinnilla. Latenssi täytyy kuitenkin huomioida, jotta uusi nauhoite voidaan synkronisoida aikaisempien kanssa. [42]

# 4. KEHITYSALUSTAT JA TEKNOLOGIAT

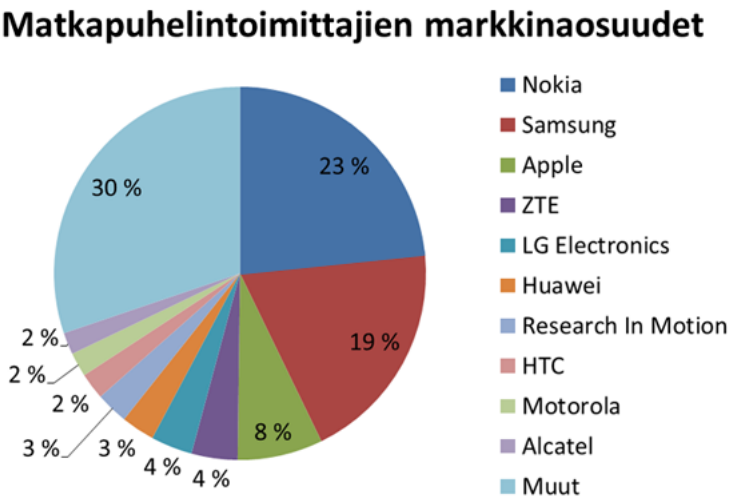
Mobiililaitteet ovat olleet jo vuosia nopean kehityksen alaisina ja uusia innovaatioita tulee markkinoille tiheään tahtiin. Kuten Steve Jobs kerran sanoi, “Aina välillä markkinoille tulee laite, joka muuttaa kaiken [12].” Eräs tällainen käänteentekevä laite oli Applen iPhone, jonka myötä kosketusnäyttöjen käyttö mobiililaitteissa lisääntyi huomattavasti. Nykyisin esimerkiksi Nokia on tuonut kosketusnäytöt myös halvempiin malleihinsa.

Erilaisia mobiilialustoja ja mobiilisovelluskehyksiä on olemassa runsas joukko. Tähän tutkimukseen on pyritty valitsemaan eniten käytettyjä alustoja ja teknologioita. Tutkimus kattaa seuraavat mobiilialustat: Nokian Series 40, Symbian[3] ja Harmattan, Googlen Android, Research In Motionin BlackBerry sekä Applen iOS. Series 40- ja BlackBerry-alustoilla käytetään Java ME -sovellusympäristöä ja Symbian- ja Harmattan-alustoilla käytössä on Qt sekä Qt Quick.




**Kuva 4.1:** Älypuhelinkäyttöjärjestelmien markkinaosuudet vuoden 2011 viimeisellä neljänneksellä. [13]

Kaaviossa 4.1 esitetty grafiikka perustuu maailmanlaajuisesti loppukäyttäjille toimitettujen älypuhelinkäyttöjärjestelmien myyntilukuihin. Symbian on pudonnut vuoden takaiselta ykköspaikaltaan kolmanneksi ja Android hallitsee nyt tilastoa selvästi. iOS on niin ikään parantanut reilusti vuoden takaisesta. Oikeastaan kaikki muut ovatkin menneet sitten alaspäin. Badan osuus on pysynyt lähes entisellään. Hieman epäselvää on mitä kaaviossa tarkoitetaan Microsoft-käyttöjärjestelmällä,




**Kuva 4.2:** Matkapuhelintoimittajien markkinaosuudet myytyjen laitteiden perusteella vuoden 2011 viimeisellä neljänneksellä. [13]

mutta oletettavasti se edustaa Windows Phone -käyttöjärjestelmää. Edes Nokian markkinoille tuomat Lumia 710 and 800 eivät ole estäneet Windows Phonen pudostusta.

Toimitettujen mobiililaitteiden määrässä Nokia on edelleen selkeä ykkönen 23,4 prosentin markkinaosuudellaan, kuten kuva 4.2 osoittaa. Kuitenkin pudotusta on tullut jonkin verran ja Samsung onkin enää vain muutamien prosenttien päässä. Apple ja ZTE tuplasivat myyntinsä, mutta ovat edelleen jokseenkin kaukana kärkikaksikosta. LG:n myynti sen sijaan lähes puolittui, kun taas Huawei, HTC ja Alcatel kirivät hieman. Research In Motionin ja Motorolan tulokset laskiva hieman.

## 4.1 Java ME

Java sai alkunsa ajatuksesta, joka suoraan suomennettuna kuuluu: "Kirjoita kerran, aja kaikkialla." Ajatus oli siis kehittää kieli, joka mahdollistaisi kertaalleen koodatun ohjelman ajamisen millä tahansa Java-virtuaalikoneen tarjoavalla alustalla. Java Micro Edition (JME, aiemmin J2ME) on standardi-Javan kevennetty versio. Se on suunniteltu pieniin laitteisiin, joiden toimintaympäristö tarjoaa esimerkiksi vain rajallisen muisitimäärän, pienen näytön ja rajoitetusti suoritustehoa. [39]

Moni Java ME:tä laitteissan käyttävä yhtiö on laajentanut kuitenkin sen ominaisuuksia tarjoamalla omia lisärajapintojaan. Vaikka Java ME mielletään helposti vain massamarkkinoille suunnattujen halpalaitteiden sovellusympäristöksi, esimerkiksi Research In Motion on omaksunut sen käytön myös älypuhelin-mallistossaan. Myös Nokian Series 40 -sarjan laitteet tukeutuvat vahvasti Java ME -ympäristöön.

Konfiguraatio on termi, jolla määritellään käytettävät Java-kielen ominaisuudet ja Java-virtuaalikoneen ydinkirjastot. Java ME jakautuu CDC- ja CLDC-

konfiguraatioihin, joista jälkimmäinen on rajatumpi ja suunnattu lähinnä matkapuhelinten ja kämmenmikrojen kaltaisille laitteille. Lisäksi Java ME:ssa on käytössä termi Profiili, joka on konfiguraation laajennus. Se tarjoaa ohjelmistokehittäjille kirjastoja, jotka mahdollistavat ohjelmoimisen tietyn tyyppisille laitteille. Esimerkiksi Mobile Information Device Profile (MIDP) määrittelee rajapinnat muun muassa käyttöliittymäkomponenteille, syötteen- ja tapahtumienkäsittelyyn sekä verkkoominaisuuksille.

## 4.2 BlackBerry

Kanadalainen Research In Motion (RIM) esitteli ensimmäisen BlackBerry-laitteensa vuonna 1999. Käyttöjärjestelmänä laitteissa on BlackBerry OS. Ohjelmat BlackBerrylle kirjoitetaan Javalla joko natiivina BlackBerry-sovelluksena tai Series 40:n tapaan Java ME:llä. Kuten Nokia, myös Research In Motion on laajentanut Java ME:n rajapintatarjoamaa omilla lisärajapinnoillaan. Research In Motionin uusimpien älypuhelinmallien käyttämä käyttöjärjestelmä on edennyt jo versioon 7.0. [34] Tähän työhön liittyvät mittaukset on tehty BlackBerry 6.0 -versiota vasten Java ME -sovelluksen avulla.

Ohjelmointiympäristönä on suositeltavaa käyttää Eclipseä yhdessä uusimpien BlackBerryn Eclipse-liitännäisten kanssa. Saatavilla on myös BlackBerryn oma Java-kehitysympäristö BlackBerry JDE. [34]

Resourch in Motion on hiljattain ilmoittanut kehittävänsä uutta BBX-alustaa BlackBerry OS:n ja hankkimansa QNX-alustan pohjalta. Uusi alusta tukee HTML5:ä yhdessä BlackBerry WebWorksin kanssa, Adobe AIR:ia sekä natiivi C:tä ja C++:aa. Lisäksi sillä voidaan ajaa Android-sovelluksia. [33] BlackBerryn Playbook-käyttöjärjestelmässä voidaan ajaa jo Qt-sovelluksia, joten C++-tuki luvannee hyvää myös BBX:n suhteen.

## 4.3 Series 40

Series 40 -alusta on maailman eniten käytetty mobiililaitteille kehitetty alusta. Juuri levinneisyytensä vuoksi Series 40 on hyvä vaihtoehto ohjelmistokehittäjälle. Se tarjoaa periaatteessa mahdollisuuden saavuttaa hyvin laaja käyttäjäkunta levittämilleen ohjelmille. Työskenteli sitten Series 40 web app -teknologioiden kanssa, Java:n tai vaikkapa Adoben Flash Liten parissa, Series 40 tarjoaa suuret markkinat ympäri maailman. Käytännössä haasteelliseksi saattaa kuitenkin muodostua ohjelmien kansainvälistämisen ja lokalisoinnin tarve. Series 40 tarjoaa Java ME -ohjelmistokehitykseen MIDP- ja CLDC-teknologiat, joiden lisäksi spesifikaatioita on laajennettu joukolla lokaatioon, kommunikaatioon, mediaan ja grafiikkapiirtoon liittyviä rajapintoja. [25]

Tässä työssä tarkastelun alaisena Series 40:ssä käytetyistä tekniikoista on ainoastaan Java ME. Java ME -ohjelmistokehitykseen usein käytettyjä ohjelmointiympäristöjä ovat Eclipse ja Netbeans. Nokia Developer -portaalista voi ladata Nokia SDK for Java -paketin, joka sisältää kehitystyökalut Series 40 -alustalle. Pakettiin kuuluu esimerkiksi emulaattori, rajapintadokumentaatio, koodiesimerkkejä sekä työkalut virheiden jäljittämiseen. [24]

## 4.4 Qt ja Qt Quick

Qt on kattava alustariippumaton C++-ohjelmistokehitysympäristö, jolla voidaan toteuttaa graafisia käyttöliittymiä. Vastaavasti kuin Javassa ajatuksena on, että ohjelmisto kirjoitetaan kerran ja sen jälkeen se on ajettavissa eri alustoilla. Qt-sovelluskehys julkaistiin Toukokuussa 1995 ja sitä on kehittäny norjalainen Trolltech. [4] Vuonna 2008 Nokia hankki Trolltechin omistukseensa [29].

Alustariippumattomuutensa ansiosta Qt-sovellukset tavoittavat suuren määrän Symbian-laitteita unohtamatta Maemoa sekä markkinoille hiljattain ilmestynyttä Nokian N9:ää, jossa käytetään MeeGo 1.2 Harmattan -alustaa. Qt-sovelluksia voidaan ajaa myös Android-laitteilla avoimen lähdekoodin Necessitas-projektin avulla. Samantyyppinen projekti voisi tulla mahdolliseksi myös Windows Phone -alustalle, jos Windows Phone 8:ssa sallitaan natiiviohjelmien ajaminen. Itseasiassa myös Windows Phone 7:ssä on onnistuttu ajamaan natiivikoodia [5], mutta Qt:ta alustalla tuskin tullaan näkemään.

Qt sisältää monipuoliset kirjastot niin työpöytä- kuin mobiilisovellustenkin toteuttamiseksi. Helpottaakseen ja nopeuttaakseen nykyaikaisten virtaviivaisten käyttöliittymien luomista on Nokia kehittänyt Qt Quick-kehyksen. Se on kokoelma teknologioita, jotka laajentvat entisestään Qt:n ominaisuuksia. Qt Quick:n yksi tärkeimmistä tekniikoista on QML. Se on CSS:ää muistuttava merkkauskieli, jonka avulla on mahdollista määritellä käyttöliittymän komponentit, niiden koot ja sijainnit, sekä erilaisia siirtymäanimaatioita. QML:ssä käyttöliittymä kuvataan hierarkkisesti puumaiseen rakenteeseen.

QML:ään voidaan kirjoittaa JavaScriptiä joko suoraan tai sisällyttämällä ulkoisia js-tiedostoja. Lisäksi QML:ään voidaa tuoda omia elementtejä C++:sta käsin. Qt-sovelluksessa hyvä tapa onkin kirjoittaa ohjelman toimintalogiikka C++:lla, erottaa käyttöliittymälogiikka QML-tiedostoihin ja toteuttaa JavaScriptillä käyttöliittymää koskevat toiminnallisuudet, kuten elementtien ja näkymien animointi, sijaintien laskeminen tai muu sellainen.

Qt SDK sisältää kaiken tarpeellisen Qt- ja QML-sovellusten toteuttamiseen kuten itse sovelluskehyksen ja käännöstyökalut. Lisäksi sovelluskehitysympäristön mukana tulee alustariippumaton Qt Creator -ohjelmointiympäristö ja Qt Designer -sovellus käyttöliittymien suunnittelua helpottamaan. Qt Creator tarjoaa muun muassa koo-

dieditorin C++:n ja JavaScriptin kirjoittamiseen, mobiilisimulaattorin, versionhallintatuen sekä työkalut virheen jäljitykseen. SDK:n mukana toimitetaan myös Qt Mobility – kokoelma rajapintoja, jotka mahdollistavat esimerkiksi kameran, sensoreiden tai GPS:n käyttämisen.

## 4.5 Symbian

Symbian OS on laajalle levinnyt mobiilikehitysalusta, jonka juuret ulottuvat 90-luvulle saakka Psion mobiililaitteisiin. Tie Psion laitteiden käyttöjärjestelmästä Symbian-älypuhelinalustaksi kulki useiden vaiheiden kautta Symbian Foundationin haltuun. Symbian Foundationin omisti alunperin Ericsson, Matsushita, Motorola, Nokia ja Psion [11]. Sittemmin Symbianin kehitys on siirtynyt Symbian Foundationilta yksinomaan Nokialle [30] ja hiljattain Nokia ulkoisti kehityksen eteenpäin Accenturelle [31]. Symbian tukeutuu C++:aan ja on vahvasti olio-orientoitunut alusta. Se pohjautuu niin sanottuun mikrokernel-arkkitehtuuriin [11].

S60, on avoin ohjelmistoalusta laitteille, jotka käyttävät Symbian OS -käyttöjärjestelmää. S60 yhdistetään usein Nokian laitteisiin, sillä alusta on Nokian omistuksessa. Se on kuitenkin käytössä myös joidenkin muidenkin valmistajien kuten Sonyn laitteissa. S60-alustalla toimivat niin Javalla, Symbian C++-sovelluskehyksellä, kuin myös Flash Litellä luodut ohjelmat. [12]

S60:n uusin versio tunnetaan nykyisin nimellä Symbian[1]. Vaikka ohjelmistokehitys Symbianille tehdäänkin pääasiassa Qt-sovelluskehystä ja Qt SDK:n tarjoamia työkaluja käyttäen, myös Symbianin natiivirajapintoja voidaan edelleen hyödyntää.

## 4.6 Harmattan

MeeGo on Intelin ja Nokian alulle panema hanke, jonka tarkoituksena oli yhdistää Moblin- ja Maemo-ohjelmistoalustat. MeeGo on Linux-pohjainen ohjelmistoalusta, joka tukee laajaa joukkoa laitearkkitehtuureja mukaan lukien taskukokoiset mobiilitietokoneet, minikannettavat, tablet-tietokoneet, mediapuhelimet, verkkoon kytkettyt televisiot sekä IVI-järjestelmät (in-vehichle infotainment). [23] Alkuvuodesta 2011 Nokia ilmoitti keskittyvänsä Windows Phone -kehitykseen ja niinpä MeeGo-hanke jäi taka-alalle ainakin Nokian osalta. Sittemmin MeeGon kehitys on lopetettu, mutta Samsung ja Intel ovat alkaneet kehittää projektin pohjalta Tizen-mobiilikäyttöjärjestelmää.

Kesäkuussa 2011 Nokia julkaisi N9-puhelimen, joka perustuu MeeGo 1.2 Harmattan alustaan. Harmattan ei kuitenkaan perustu suoranaisesti MeeGo-projektiin, vaan se on jatkokehitetty Maemo 5 -alustan pohjalta. Se on kuitenkin MeeGo-yhteensopiva. N9 on ainakin toistaiseksi ainut markkinoilla oleva Harmattan-laite. Lisäksi Nokia jakoi ohjelmistokehittäjille Qwerty-näppäimistöllä varustettuja N950-

laitteita, jonka alustana on myös Harmattan. Kehitys Harmattanille tapahtuu Qt:lla ja Qt Quick:llä.

## 4.7 iOS

Apple esitteli ensimmäisen iPhone-puhelimensa vuoden 2007 alussa. Se edusti uudenlaista matkapuhelinta, jossa yhdistyivät puhelin, iPod, kosketusnäyttö ja kehittyneet internet-ominaisuudet, kuten sähköposti, internet-selailu, hakeminen ja kartat yhdessä käteen sopivassa laitteessa. [17] iPhone oli selvästi vallankumouksellinen laite ja kiistatta eräs mobiiliteollisuuden virstanpylväistä. Esimerkiksi iPhone 3G valtasi Yhdysvaltain mobiilimarkkinat alle neljässä kuukaudessa syrjäyttäen pitkään myydyimmän mobiililaitteen paikkaa pitäneen Motorolan RAZR-puhelimen. [12] iPhonen vallatessa älypuhelinmarkkinoita, myös ohjelmistopuolella Apple viitoitti tietä. Mobiilisovellusten suosio kasvoi valtaisaksi App Storen myötä ja kuudessa kuukaudessa iPhone 3G:n julkaisemisen jälkeen ohjelmalatauksia oli jo yli 300 miljoonaa. [12]

iOS tunnettiin aiemmin nimellä iPhone OS. Applen iOS:ssä yhdistyvät käyttöjärjestelmä sekä teknologiat, joita käytetään natiivisovellusten ajamiseen laitteilla kuten iPad, iPhone ja iPod touch. Vaikkakin sillä on paljon yhteistä Mac OS X -käyttöjärjestelmän kanssa, se on suunniteltu mobiilikäyttöympäristön tarpeita varten. [3]

Ohelmistokehitys iOS:lle tapahtuu käyttäen Applen omaa integroitua ohjelmointiympäristöä nimeltä Xcode. Se tarjoaa työkalut niin käyttöliittymän suunnitteluun, kuin koodin kirjoittamiseen. Xcoden mukana tulee myös iOS-simulaattori, jolla ohjelmiaan voi testata ennen laitteessa ajoa. [3]

## 4.8 Android

Google Inc., T-Mobile, HTC, Qualcomm, Motorola ja joukko muita teknologiayhtiöitä julkistivat uuden mobiilialustan nimeltä Android vuoden 2007 loppupuolella. Sen kehityksestä vastaa monikansallinen Open Handset Alliance -organisaatio. [28] Android on avoimeen lähdekoodiin perustuva ohjelmistopino, johon kuuluu muun muassa käyttöjärjestelmä, väliohjelmistot ja perus puhelinsovellukset. Lisäksi Android tarjoaa joukon rajapintakirjastoja, jotka mahdollistavat monipuolisten mobiilisovellusten kirjoittamisen. [14] Avoimuudella haluttiin lisätä ohjelmistokehittäjien välistä yhteistyötä ja kiihdyttää siten tahtia, uusien käyttäjiin vetoavien mobiilipalveluiden tuottamiseksi. [28]

T-Mobile julkaisi ensimmäisen Android-puhelimen syyskuussa 2008. Samaan aikaan Google kehitti kuumeisesti Android-sovelluksia, että esimerkiksi hakukone-, sähköposti- sekä karttapalvelut saataisiin tarjolle myös Androidille, niin kuin ne oli-

vat jo tarjolla iPhonelle, BlackBerrylle, Windows Mobile -laitteille ja monille muille laitteille [40]. Google toi oman Nexus One -Android-puhelimensa markkinoille vuoden 2010 alussa [38].

Android-ohjelmat kirjoitetaan Javalla sekä käyttöliittymän kuvaukseen tarkoitetulla XML-pohjaisella kuvauskielellä. Eräs Androidin tärkeimmistä elementeistä on Dalvik-virtuaalikone. Dalvik hyödyntää Linux-kerneliä matalan tason toiminnallisuuden kuten tietoturvan, säikeistyksen sekä prosessin- ja muistinhallinan hoitamiseen. On myös mahdollista kirjoittaa C/C++-ohjelmia suoraan Linux-käyttöjärjestelmän päälle, mutta useimmiten se ei ole tarpeen. [22]

Ohjelmien kehittämiseksi Androidille ohjelmoija tarvitsee Android SDK:n sekä Java Development Kitin. Lisäksi Java ohjelmointiympäristön kuten Eclipsen. Sekä Android SDK, JDK, että Eclipse ovat saatavilla niin Windowsille, kuin myös MacOS:lle ja Linuxille. SDK-työkalut ja emulaattori toimivat niin ikään kaikilla kolmella käyttöjärjestelmällä. [22] Eclipseä käytettäessä on suotavaa myös asentaa Android Development Tools -liitännäinen, joka helpottaa muun muassa uuden Android-projektin aloittamista, käyttöliittymien luomista ja virheiden jäljittämistä. [14]

## 4.9 Windows Phone

Microsoft lähti mukaan PDA (Personal Digital Assistant) markkinoille esiteltyään Pocket PC tuotemerkkinsä vuonna 2000. Kaksi vuotta myöhemmin Microsoft julkaisi ensimmäisen älypuhelimensa, jonka yhteydessä käytettiin tuotemerkkiä Smartphone. Myöhemmin nämä kaksi linjaa kuitenkin yhdistyivät ja saivat käyttöjärjestelmäkseen Windows Mobilen. [10] Se pohjautui Windows CE -kerneliin ja Win32-rajapintaa vasten kirjoitettuja ohjelmia pystyikin ajamaan useimmilla Windows Mobile -laitteilla. [12]

Windows Phone on Windows Mobilen seuraaja, joskaan se ei ole taaksepäin yhteensopiva. Siinä missä Windows Mobile oli enemmän suunnattu yrityskäyttöön, Windows Phone on suunniteltu nimenomaan iPhonen ja Androidin kilpailijaksi kuluttajamarkkinoille. Sen suunnittelussa on lähdetty puhtaalta pöydältä ja lopputulos erottuukin tuoreudellaan muista alustoista. [19] Vaikka Windows Phone 7:n ensimmäisestä versiosta puuttui vielä joitakin tärkeitä ominaisuuksia, se oli silti hyvin pitkälle viimeistelty ja pätevän oloinen alusta. Windows Phonen eräs tärkeimmistä uudistuksista on varta vasten kosketusnäyttöjä ajatellen suunniteltu Metro UI, joka poikkeaa melkoisesti Microsoftin totutusta linjasta. Tyyli on pyritty pitämään mahdollisimman selkeänä ja minimalistisena menettämättä kuitenkaan tyylikkyyttä. Metro-tyyli on sittemmin ulotettu myös Windows 8 -käyttöjärjestelmään, joka on suunnattu työpöytätietokoneiden ohella tablet-laitteissin.

Microsoft ei kehittänyt Windows Phonea varten uusia ohjelmointikieliä tai so-

velluskehyksiä, vaan hyödynsi yhtiön olemassa olevia teknologioita kuten C# ja .NET. [20] Kehitysympäristönä käytetään Visual Studiota ja lisäksi ohjelmien kehittäminen tapahtuu joko SilverLightia- tai XNA-pelikehystä käyttäen. Microsoft on pystynyt pitämään oppimiskäyrän matalana, sillä kyseisille teknologioille oli jo olemassa laaja käyttäjäkunta. [19]

Sovelluskehitykseen tarvitaan myös Windows Phone SDK sekä Zune-ohjelmisto laitteiden kytkemiseksi tietokoneeseen. SDK:n mukana tulee Windows Phone -emulaattori ja Zunen avulla ohjelmia voidaan ajaa myös suoraan laitteessa. Kehitystyökalujen mukana toimitetaan lisäksi Microsoft Expression Blend, joka on tarkoitettu esimerkiksi käyttöliittymäsuunnitteluun ja siirtymäanimaatioiden määrittämiseen.

# 5. KIRJASTOT JA RAJAPINNAT

Moni alusta toimii niin sanottussa hiekkalaatikossa, jolloin ohjelmistokehittäjä joutuu monesti tyytymään pelkästään alustan tarjoamiin kirjastoihin ja rajapintoihin. Joillekin alustoille on kuitenkin mahdollista kirjoittaa vaikka alusta alkaen oma audiokirjasto, jos alustan omat kirjastot eivät tunnu riittäviltä. Yleisesti ottaen eri alustojen äänikirjastot ovat varsin kattavia ja verkosta löytyy lisäksi useita avoimeen lähdekoodiin perustuvia äänikirjastoprojekteja. Välillä on jopa hankala hahmottaa mikä olisi sopivin kirjasto omaan käyttötarkoitukseen laajan tarjonnan vuoksi. Esimerkiksi Androidille, iOS:lle ja Harmattanille on käytettävissä jokaiselle ainakin neljä eri audiorajapintaa.

## 5.1 BlackBerry

BlackBerry on pitkään tukeutunut Java ME:n ympärille. Niinpä audion toistossa käytetään Java ME:n tarjoamaan Mobile Media -rajapintaa (MMAPI/JSR-135). Se on suhteellisen helppokäyttöinen, joskin audiokontrollin suhteen melko rajoittunut. BlackBerryllä audiota on myös mahdollista toistaa selaimen avulla, mutta tämä lähetymistapa sovelttuu paremmin pitkien äänileikkeiden toistamiseen.

## 5.2 Series 40

Series 40:llä ei audiorajapinnan valitseminen tuota harmaita hiuksia - vaihtoehtoja on nimittäin tasan yksi. Käytettävissä on BlackBerryn tapaan Mobile Media API. Audion ohella se tarjoaa muun muassa videontoistollisia ominaisuuksia sekä mahdollistaa kameran käytön. Rajapinta on käytettävissä myös Symbian laitteilla, mutta käytännössä Java ME -sovelluksia harvemmin enää suunnitellaan Symbian-ympäristöissä käytettäväksi.

## 5.3 Symbian

Symbian-alustalla Qt-kehittäjän käytettävissä ovat QtMobilityn tarjoamat audiorajapinnat sekä Symbianin Multi Media Framework -natiivirajapinnat. Multi Media Framework sisältää korkean tason audiorajapinnan, mutta mahdollistaa pääsyn myös Symbianin matalimman tason DevSound-audiorajapintaan.

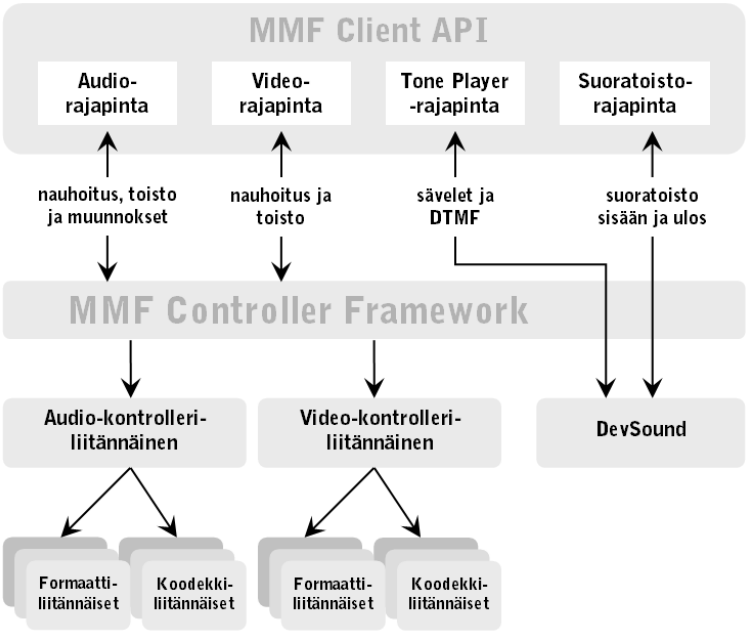
### 5.3.1 QtMultimediaKit

QtMultimediaKit tarjoaa joukon rajapintoja multimediatoiminnallisuuksien toteuttamiseksi Qt:ssa. Se sisältää esimerkiksi luokkia, joiden avulla kehittäjä voi nauhoittaa ja toistaa audiota, käyttää kameraa, sekä hallita mediasisältöisiä kokoelmia. Se korvaa aikaisemmin käytetyn Phonon-rajapinnan. Vaikka Phonon on vielä saatavilla, ei sen käyttöä varsinaisesti suositella. [26]

Multimedia QML Plugin tuo QtMultimediaKitin audio-ominaisuudet QML:ään. Kirjasto sisältää esimerkiksi helppokäyttöiset audio- video- sekä kamera-elementit. Ihan kaikkeen QML ei ainakaan vielä taivu, eikä ole syytäkään. Esimerkiksi raakaaudion käsittely on järkevää tehdä edelleen Qt-kerroksessa.

### 5.3.2 Multi Media Framework Client API

Multi Media Framework on Symbianin natiivi audiorajapinta. Se sisältää rajapinnat audion ja videon toistoon, audion suoratoistoon, sekä mahdollistaa sävelten generoimiseen. Multi Media Framework pitää sisällään myös kevyen sovelluskehyksen, joka mahdollistaa ominaisuuksien laajentamisen liitännäisillä. [24]

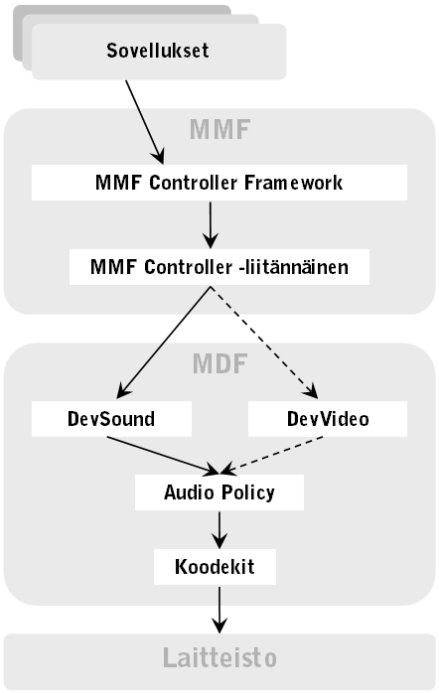



**Kuva 5.1:** Multi Media Framework

### 5.3.3 DevSound

DevSound API tarjoaa rajapinnan Symbian OS:n ja audiolaitteiston välillä. Sen tehtävänä on muun muassa audion toistaminen ja nauhoittaminen [24] DevSound on Symbianin matalimman tason audiorajapinta. Se on laitteisto-riippuvainen, joten

se ei välttämättä ole käytettävissä kaikilla Symbian-alustoilla. DevSoundia käytetään CMMFDevSound-nimisen luokan kautta ja sen avulla saavutetaan ylemmän tason rajapintoja tarkempi kontrolli audiolaitteistoon sekä -puskuriin. [6]




**Kuva 5.2:** DevSound

### 5.3.4 Qt GameEnabler

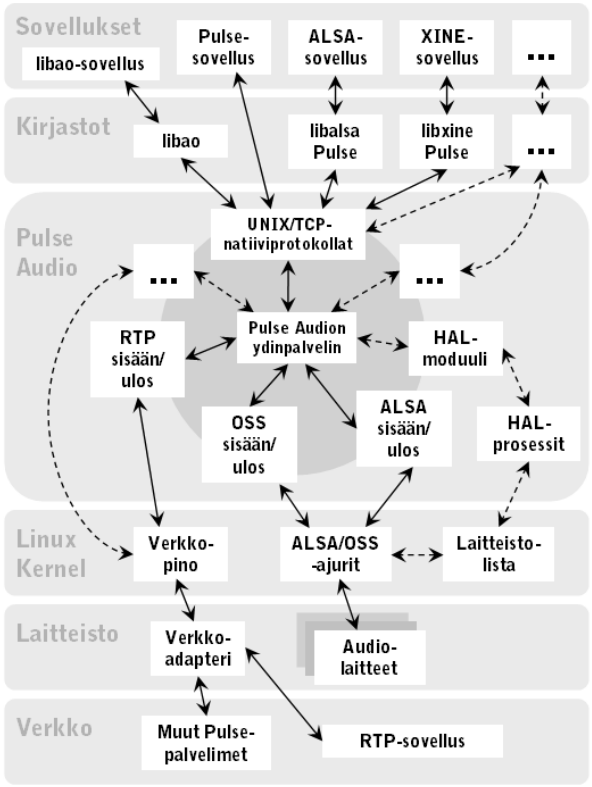
Qt GameEnabler on Nokian tuottama sovelluskehys, jonka tarkoituksena on helpottaa pelikehitystä Qt:lla. Se perustuu avoimeen lähdekoodiin, joten sitä voi vapaasti käyttää, kopioida sekä muokata. Qt GameEnabler on suunnitteltu käytettäväksi nimenomaan mobiililaitteissa. Luokkakirjaston sisältämät työkalut tehostavat OpenGL:n käyttöä Qt-sovelluksissa ohittamalla Qt:n oman renderöinnin ja korvaamalla sen natiivilla OpenGL:llä. Pelikehityksen kannalta olennainen ominaisuus on myös päivitysfunktio, joka suoraviivaistaa pelisilmukan luomista. Lisäksi kehys pitää sisällään joukon muita pieniä kehitystä helpottavia ominaisuuksia. [27]

Qt GameEnabler sisältää myös oman kevyen audiokirjastonsa, joka tarjoaa muun muassa miksaustoiminnallisuuden. Sillä ei ole riippuvuuksia kehyksen grafiikkaluokkiin, joten audiokehystä voidaan käyttää täysin itsenäisesti. Koko luokkakirjastoa ei siis ole välttämätöntä sisällyttää omaan projektiin. Kirjoitushetkellä ainoa mahdollinen ulostuloformaatti on kaksikanavainen 16-bittinen PCM, jonka näytteenottotaajuus on 22050 Hz. Audiorajapinta on edelleen kuitenkin aktiivisen kehityksen alaisena, joten parannuksia lienee luvassa. [27]

## 5.4 Harmattan

Harmattan toimii Linuxin päällä. Linuxille on tarjolla runsas valikoima avoimeen lähdekoodiin perustuvia hyvinä pidettyjä audiokirjastoja ja ajureita kuten ALSA, GStreamer sekä JACK. Niihin ei kuitenkaan juuri ole tutustuttu tämän työn puitteissa. Sen sijaan huomio kiinnittyi PulseAudio-kirjastoon, jota käytetään Harmattan-laitteissa Qt:n alla.

**PulseAudio** on järjestelmäriippumaton äänipalvelin. Se toimii muun muassa Windowsissa, OS X:llä sekä Linuxilla. PulseAudio toimii taustaprosessina, ottaen vastaan syötteet eri äänilähteistä ja ohjaten ne eteenpäin esimerkiksi äänikortille tai verkon yli toiselle PulseAudio-palvelimelle kuvan 5.3 osoittamalla tavalla. PulseAudio on avointa lähdekoodia ja sitä voi hyödyntää GNU Lesser General Public License -ehtojen mukaisesti omissa projekteissaan. [32]

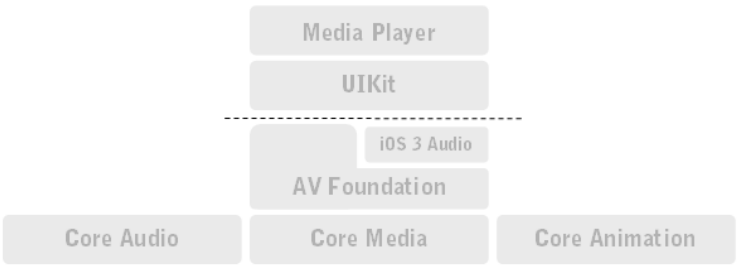



**Kuva 5.3:** PulseAudion toiminnallinen vuokaavio

PulseAudio tarjoaa kaksi lähestymistapaa: yksinkertaisen synkronisen rajapinnan ja monimutkaisemman asynkronisen rajapinnan. [24] Jos on tarve käyttää PulseAudiota suoraan, asynkroninen rajapinta lienee se mitä ollaan hakemassa.

## 5.5 iOS

iOS tarjoaa runsaan valikoiman työkaluja audiosovellusten käyttöön. Työkalut on järjestetty sovelluskehyksiksi niiden sisältämien ominaisuuksien mukaan. Vaihtoehtoja ovat Media Player -, AV Foundation -, Audio Toolbox -, Audio Unit -, sekä OpenAL-sovelluskehykset. Monipuolisen valikoiman vuoksi saattaa olla hieman hankala hahmottaa mikä kehys sopisi parhaiten omaan käyttöön. Kuva 5.4 valottaa hieman sovelluskehysten hierarkkiaa.




**Kuva 5.4:** iOS audiosovelluskehykset

### 5.5.1 Media Player

Media Player on tarkoitettu lähinnä musiikkin, videon, äänikirjojen ja podcast-lähetysten toistamiseen. Sen avulla voidaan toistaa äänipohjaista mediaa myös käyttäjän iPod-kirjastotsta. Sen avulla käyttäjälle voidaan helposti mahdollistaa erilaisten ulostuloreititysten tekeminen esimerkiksi AirPlay-ominaisuutta tukeviin laitteisiin. Rajapinnan avulla on myös mahdollista esimerkiksi toistaa videovirtoja.

### 5.5.2 AV Foundation

AV Foundation tarjoaa helposti lähestyttävän Objective-C-rajapinnan audion ja videon toistamiseen ja nauhoittamiseen. Sen avulla voi tutkia, luoda, muokata ja pakata mediatiedostoja. Medialaitteistolta tulevia virtoja voidaan myös muokata reaaliaikaisesti ennen toistoa.

### 5.5.3 Core Audio

Core Audio on kirjasto, joka pitää sisällään kattavan kokoelman C-rajapintoja audion ja multimedian toistamiseen, nauhoittaminen, muokkaamiseen, pakkaamiseen ja purkamiseen, MIDI-prosessointiin, signaalinkäsittelyyn sekä audion syntetisointiin. Tässä työssä huomio kiinnittyy Core Audion sisältämään Audio Toolbox -kehykseen sekä Audio Unit -kehykseen – tarkemmin ottaen RemoteIO-audioyksikköön. Lisäksi Core Audio tarjoaa pääsyn OpenAL-rajapintaan, joka tarjoaa ominaisuuksia pääasiassa pelikehityksen tarpeisiin.

**Audio Toolbox** tarjoaa välineet muun muassa audion tallentamiseen sekä toistamiseen synkronoidusti, saapuvien audiopakettien käsittelyyn, äänivirtojen parsimiseen ja formaattien muuntamiseen. iOS:ssä sovelluskehys tarjoaa myös rajapinnan audiosessioiden hallinnoimiseen. Yksinkertainen ääniefektin toistaminen onnistuu sillä likimain yhtä helposti kuin AV Foundationia käytettäessä.

Audio Toolbox sisältää myös Audio Queue Services -rajapinnan, jonka avulla on mahdollista suorittaa puskuroitua raakadatan suoratoistoa toteuttamalla Audio Queue -takaisinkutsufunktio. Myös tallentaminen onnistuu jonojen avulla.

**Audio Unit** on plugin-kehys, johon voidaan liittää tietyn ominaisuuden tarjoavia audioyksiköitä. Yksiköillä voidaan toteuttaa erilaisia suotimia, lisätä kaikua, tehdä formaattimuunnoksia, miksata audiovirtoja sekä tietysti toistaa audiota. [3] RemoteIO on yksikkö, jonka avulla ulosmenevän audio latenssi on mahdollista saada erityisen pieneksi, koska se toimii hyvin lähellä audiolaitteistoa. Audion toistamiseksi sovelluksen täytyy toteuttaa renderöintitakaisinkutsu, jolle audiodataa tarjoillaan sopivissa palasissa. [41] Kyseessä on siis suoratoisto.

**OpenAL** on monipuolinen alustariippumaton audiorajapinta, joka soveltuu hyvin pelisovelluksiin. Se tarjoaa erilaisia efektejä, kuten äänilähteiden mallintamisen 3D-tilassa ja Doppler-mallinnuksen. [8] Apple on sisällyttänyt OpenAL-rajapinnan jo aikaisessa vaiheessa iOS:iin. [3] Kirjastoon ei kuitenkaan tutustuttu lähemmin tämän työn puitteissa.

## 5.6 Android

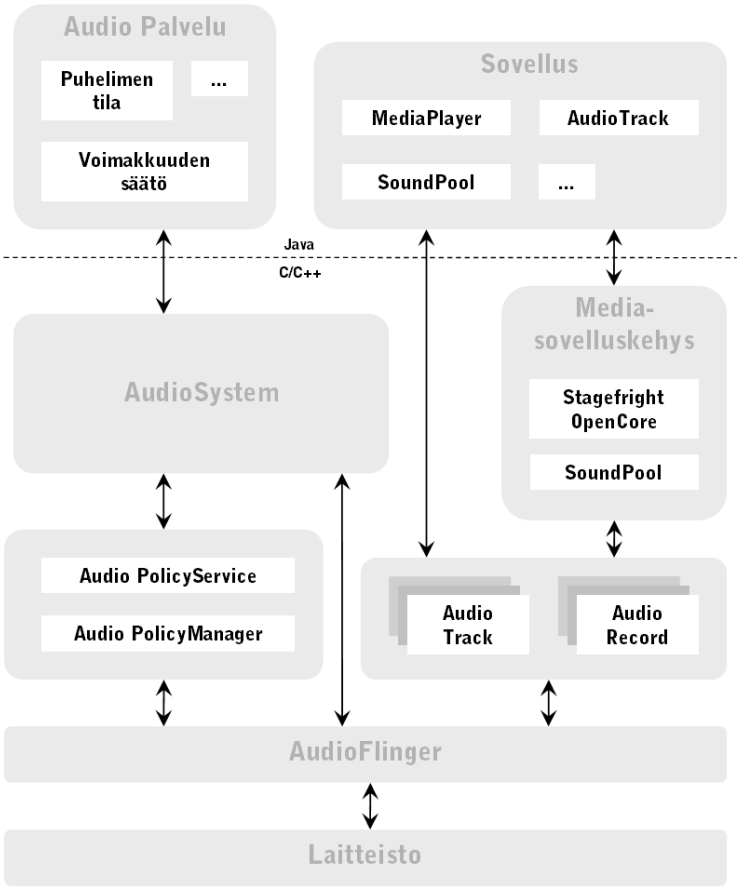
Androille on tällä hetkellä tarjolla ainakin seuraavat neljä audiorajapintaa: MediaPlayer, SoundPool, AudioTrack ja OpenSL. MediaPlayer ja SoundPool ovat helppokäyttöisempiä korkean tason rajapintoja, kun taas AudioTrack mahdollistaa myös suoratoiston. OpenSL on natiivi C++-rajapinta, joka niin ikään mahdollistaa suoratoiston, mutta on toisaalta melko hankala käyttää. [14] Kuva 5.5 havainnollistaa Andrdoidin audionkäsittelyä C/C++-kerroksessa.

### 5.6.1 MediaPlayer

MediaPlayer lienee Androidin eniten käytetty rajapinta. Se muistuttaa toiminnaltaan Java ME:n Mobile Media API:a. Se on hyvin suoraviivainen käyttää ja sopii hyvin vähemmän vaativiin käyttötapauksiin, kuten merkkiäänen toistamiseen sovelluksessa. Se on myös varsin hyvin dokumentoitu.

### 5.6.2 AudioTrack

AudioTrack on Androidin matalimman tason audiorajapinta, jonka avulla toistettavan audion perusominaisuuksiin voidaan vaikuttaa melko vapaasti. Myös puskurin




**Kuva 5.5:** Androidin audionkäsittely

koko on säädettävissä. Rajapinta tarjoaa niin sanotun "kanavan" (channel), johon voidaan puskea raakaa audiodataa, mikä periaatteessa mahdollistaa latenssin minimoimisen. AudioTrackin kohdalla on tosin raportoitu joitankin hankalia ongelmia liittyen nopeaan audion uudelleentoistoon. [15]

### 5.6.3 SoundPool

SoundPool mainitaan usein, kun tavoitellaan alhaista latenssia Androidilla. Sitä käytetään esimerkiksi ääniefektien toistamiseen peleissä silloin, kun useita ääniä täytyy pystyä toistamaan yhtäaikaa. SoundPool tosin rajoittaa toistettavien ääniraitojen määrää pysäyttämällä pienimmällä prioriteetilla varustetun äänen.

### 5.6.4 OpenSL

OpenSL on alustariippumaton C++-rajapinta, joka on suunniteltu erityisesti mobiililaitteita varten. Sen käyttämiseksi ohjelmoija tarvitsee Android NDK:n, jolla natiivikoodi käännetään .so-päätteisiksi tiedostoiksi. Natiivikoodin funktioita voidaan kutsua JNI:n (Java Native Interface) avulla Android-Java-sovelluksesta käsin.

OpenSL on sukua OpenAL-audiorajapinnalle, mutta se on kuitenkin suunniteltu uudelleen mobiililaitteita ja muita rajallisilla resursseilla varustettuja laitteita ajatellen, kun taas OpenAL on tarkoitettu alunperin työpöytäympäristöön. Onkin syytä huomata, että niiden rajapinnat eivät ole yhteensopivia. [21]

## 5.7 Windows Phone

Windows Phone -alustalla audion toistoon on käytännössä tarjolla yksi kirjasto – **XNA:n Audio Framework**. Silverlightissa on tarjolla myös MediaElement, mutta se ei ole niinkään suunnattu pelikäyttöön, vaan pikemminkin esimerkiksi taustamusiikin soittamiseen. Jos siis halutaan toistaa audiota Windows Phone -sovelluksessa on selvintä sisällyttää projektiin XNA:n audio-kirjastot myös Silverlight-sovelluksessa.

Audiorajapinnan käyttö on hyvin yksinkertaista. Ladataan äänileike ja pyydetään rajapintaa toistamaan se. Rajapinta tukee myös suoratoistoa, joten reaaliaikainen audion tuottaminen on myös mahdollista.

# 6. MITTAUSJÄRJESTELYT

Jotta voidaan vertailla eri mobiilialustojen audiolatensseja, tulee ensiksi kaikille tutkittaville alustoille toteuttaa ohjelma, joka reagoi kosketukseen soittamalla äänileikkeen. Tähän riittää hyvin minimaalinen toteutus, joka käyttää alustan tarjoamia rajapintoja hyväkseen. Minimitoteutusten lisäksi monilla alustoilla on mahdollista ottaa käyttöön ulkoisia audiokirjastoja, joten työssä myös kokeillaan erilaisia konfiguraatioita latenssien pienentämiseksi.

## 6.1 Mittausmenetelmä

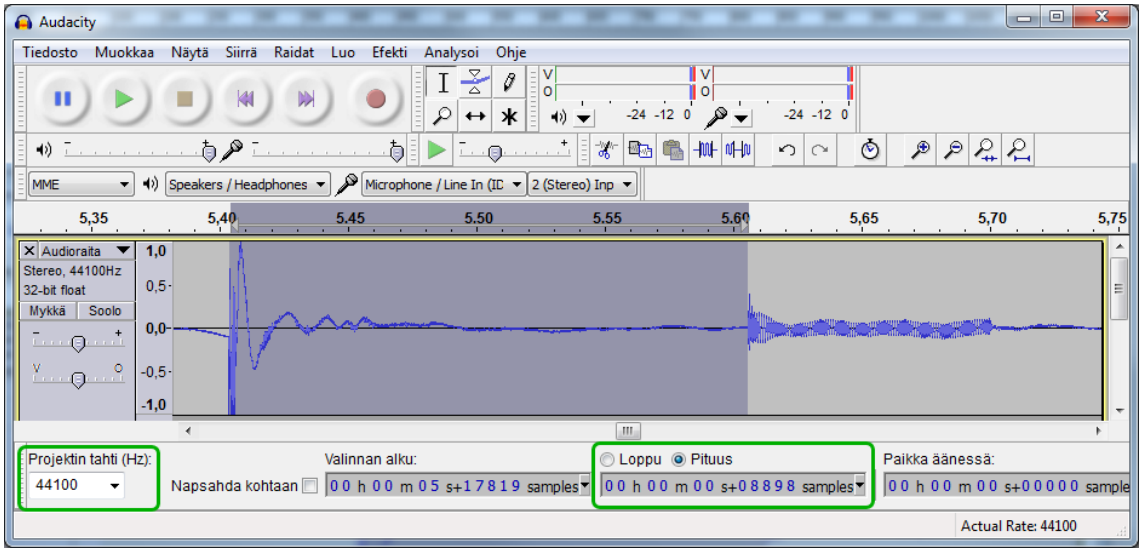
Latensseja täytyy mitata konsistentisti, jotta voidaan kerätä vertailukelpoista dataa. Mitattava latenssi koostuu kosketustapahtuman käsittelyyn kuluvan ajan ja audion toistamiseen kuluvan ajan lisäksi mahdollisista muista suoritinaikaa vievistä toimenpiteistä. Niinpä laitteiden taustaprosessit, kuten WLAN-toiminnot, on kytkettävä pois päältä mahdollisuuksien mukaan, että suoritinta kuormittavien tehtävien vaikutus mittauksiin voidaan minimoida. Täysin taustaprosessien häiriövaikutusta ei voida poistaa, joten mittauksia on tehtävä riittävä määrä, että voidaan varmistua tulosten luotettavuudesta.

Kosketusviiveen ja audiolatenssin erottaminen on hankalaa. Audion toiston viivettä voitaisiin mitata ohjelmasta käsin esimerkiksi aikaleimoja käyttämällä siten, että tallennetaan aikaleima juuri ennen kuin pyydetään äänikirjaston rajapintaa toistamaan ääninäyte ja toinen aikaleima, kun äänileikkeen toistaminen lopetetaan. Latenssi saataisiin laskettua vähentämällä ääninäytteen pituus ja toiston aloitusajankohta toiston loppumisajankohdasta. Loppumisajankohdan määrittäminen voisi onnistua sopivan tapahtumankuuntelijan avulla, jos audiorajapinta sellaista tukisi. Emme kuitenkaan voisi tietää millainen on takaisinkutsujen oikea-aikaisuudesta. Saatu data ei näin ollen olisi välttämättä vertailukelpoista eri alustojen välillä.

Myös kosketusviiveen mittaamisessa on omat haasteensa. Eräs mahdollinen tapa voisi olla suurnopeuskameran käyttäminen. Käytettävän ohjelman pitäisi antaa visuaalinen viesti laitteen näytöllä, kun kosketus on havaittu. Kuvaamalla kosketustapahtuma, voitaisiin määrittää todellisen kosketuksen ja sen rekisteröitymisen välinen viive. Tämän jälkeen kokonaisviiveestä voitaisiin karkeasti arvioida audiolatenssin suuruus, jos huomioitaisiin näytön virkistämiseen kuluva aika sekä oletettaisiin, että taustaprosessien aiheuttamat viiveet olisivat häviävän pieniä.

Kuitenkin täytyy muistaa, että lopulta tärkeintä on käyttäjän kokemus latenssista. Niinpä vain kokonaislatenssilla on todellista merkitystä, kun halutaan todentaa onko jollekin tietylle alustalle järkevä toteuttaa latenssikriittisiä sovelluksia. Kokonaislatenssin mittaaminen on onneksi kohtuullisen vaivatonta, joskin se vaatii jonkin verran manuaalista työtä.

Kokonaislatenssi voidaan mitata käyttämällä tavallista mikrofonia seuraavasti: kosketetaan näyttöä napakasti siten, että kosketus voidaan kuulla, ja nauhoitetaan myös kosketustapahtumaan liitetyn ääninäytteen soiminen. Kosketusäänen ja ääninäytteen välinen viive voidaan määrittää lähes millä tahansa äänieditorilla, joka osaa näyttää nauhoitteen aaltomuodon. Toistettavan ääninäytteen tulee olla muodoltaan jyrkkäreunainen, jotta näytteen alkuhetki voidaan nähdä helposti. Kuvasta 6.1 nähdään miten latenssin määrittäminen tapahtuu Audacityn avulla. Audacity osaa näyttää valinnan pituuden suoraan millisekunteina, mutta tarkempaan tulokseen päästään jakamalla näytteiden määrä näytteenottotaajuudella. Kuvan mittauksen latenssiksi saataisiin laskutoimituksella 8898 / 44100 Hz noin 202 ms, kun sen sijaan Audacity ilmoittaisi valinnan pituudeksi tasan 200 ms.




**Kuva 6.1:** Latenssin määrittäminen Audacityllä

## 6.2 Testilaitteisto

Jotta mittaustuloksia voidaan vertailla, olisi myös suotavaa, että käytettävät mobiililaitteet ovat tehokkuudeltaan samalla viivalla. Toisaalta ei ole olemassa tehokkuudeltaan täysin yhteneviä verrokkilaitteita eri alustoille. Etenkään Series 40 -alustalle ei ole laitetta, joka voisi kilpailla vaikkapa iPhonen kanssa samassa luokassa. Toisaalta iOS:n yhteydessä laitevalikoima on hyvin suppea, joten vaihtoehtoja ei juuri ole. Lisäksi esimerkiksi Windows Phone 7 määrittelee minimivaatimukset myytäville

laitteille, joten kovin heikkotehoisia laitteita ei markkinoilta ei edes löydy.

Koska eri alustoja on hankala vertailla täysin yhtenevillä laitekonfiguraatioilla ja tämän työn puittessa käytössä on vain rajattu laitekanta, onkin laitevalinnoissa tehtävä kompromisseja. Ei ole myöskään täysin selvää mitkä laitteiston osat ovat itseasiassa audiolatenssin kannalta oleellisimmat. Laitteiston tärkein osa äänentoistoa ajatellen on periaattessa käytettävä äänipiiri. Kuitenkin äänipiirin tehtävä on lähinnä audiosignaalin muuntaminen digitaalisesta analogiseksi. Tämän toimenpiteen aiheuttama latenssi on hyvin pieni (luokkaa 1-2 ms [2]). Toisaalta, jos äänipiirin käyttämän puskurin koko on kiinteä, olisi se latenssin kannalta hyvin mielenkiintoinen tieto. Äänipiirien tarkkoja spesifikaatioita ei ole kovin helposti saatavilla, joten niihin laitevalintoja on vaikea perustaa. Joillekin testilaitteille löytyi äänipiireistä epävirallisia tietoja, jotka on saatu purkamalla laitteet. Tietoja on listattu taulukossa 6.1. Näiden tietojen pohjalta ei kuitenkaan voida luotettavasti sanoa miten paljon laitteen audion toistosta todellisuudessa on mainitun piirin vastuulla. Laitevalmistajat näyttävät suosivan uudemmissa malleissaa niin sanottuja järjestelmäpiirejä (system on chip), joissa suurin osa laitteen keskeisistä toiminnoista on integroitu yhdeksi piiriksi. Esimerkiksi Qualcommin Snapdragon-piiriperhe on hyvä esimerkki tällaisista mikropiireistä. Ne sisältävät muun muassa näytönohjainpiirin, suorittimen ja signaaliprosessorin.

**Taulukko 6.1:** Mittauksiin käytetty testilaitteisto

| Laite | Prosessori | Muisti | Äänipiiri | Multi-touch | Näyttö |
|---|---|---|---|---|---|
| Nokia X3-02 | | 64 Mt | ? | Ei | Resistiivinen |
| Sony Ericsson Satio | 600 MHz | 256 Mt | ? | Ei | Resistiivinen |
| Nokia XM 5800 | 434 MHz | 128 Mt | Toshiba DAC-33 | Ei | Resistiivinen |
| Nokia N8 | 680 MHz | 265 Mt | Broadcom BCM2727 | Kyllä | Kapasitiivinen |
| Nokia 701 | 1 GHz | 512 Mt | Broadcom BCM2763 | Kyllä | Kapasitiivinen |
| Nokia N9 | 1 GHz | 1 Gt | Texas Instruments TLV320DAC3100 | Kyllä | Kapasitiivinen |
| Samsung Nexus S | 1 GHz | 512 Mt | Wolfson Microelectronics WM8994 | Kyllä | Kapasitiivinen |
| ZTE Blade | 600 MHz | 512 Mt | Qualcomm MSM7225 | Kyllä | Kapasitiivinen |
| Apple iPhone 4 | 1 GHz | 512 Mt | Cirruc Logic CLI1560B0 | Kyllä | Kapasitiivinen |
| BlackBerry Torch 9800 | 624 MHz | 512 Mt | Texas Instruments TLV320AIC36IZQER | Kyllä | Kapasitiivinen |
| BlackBerry Torch 9850 | 1,2 GHz | 768 Mt | ? | Kyllä | Kapasitiivinen |
| Samsung Omnia 7 | 1 GHz | 512 Mt | Qualcomm QSD8250 Snapdragon | Kyllä | Kapasitiivinen |
| HTC HD7 | 1 GHz | 576 Mt | Qualcomm QSD8250 Snapdragon | Kyllä | Kapasitiivinen |
| Nokia Lumia 800 | 1,4 GHz | 512 Mt | Qualcomm MSM8255 Snapdragon | Kyllä | Kapasitiivinen |

Myös prosessoriteholla on jonkin verran vaikutusta latenssiin. Prosessoriteholla voidaan jossain määrin kompensoida alustan mahdollisesti heikkoja arkkitehtuurivalintoja ja siksi olisikin hyvä mitata latensseja myös muilla kuin valmistajien lippulaivamalleilla. Toisaalta todennäköisemmin pullonkaula lienee laiteajurissa, jolloin tärkeää olisi myös valita samantasoisia laitteita monelta eri valmistajalta.

Kovin kattavaan otokseen ei tämän tutkimuksen puitteissa kuitenkaan ole resursseja, eikä työssä pyritäkään selvittämään mitkä komponenttivalinnat vaikuttavat latenssiin. Ohjelmoija on lopulta kuitenkin täysin laitevalmistajien armoilla ja voi itse vaikuttaa latenssiin ainostaan koodista käsin valitsemalla käyttötarkoitukseen sopivan kirjaston, näytteenottotaajuuden sekä puskurikoon. Täytyy myös muistaa, että eniten kuluttajien käyttämiä alustoja eivät suinkaan ole uusimmat julkaisut, vaan

**Kuva 6.2:** Mittauksiin käytetty testilaitteisto

jo pidempään markkinoilla olleet vaihtoehdot. Tässä työssä keskitytään pääosin tarkastelemaan alustojen nykyhetken tilannetta ja tasaisin laitteistolinjaus saadaan, kun valitaan 1 GHz:n suorittimella valittuja laitteita. Toinen tasainen teholuokka tuntuisi olevan 600 MHz:n kellotaajuutta käyttävät monesti vähän vanhemmat laitteet.

## 6.3 Ääniformaatit

Ääniformaatin valinta vaikuttaa omalta osaltaan latenssiin. Jos ääninäyte käyttää kompressoitua eli pakattua formaattia, se joudutaan ensiksi purkamaan raa'aksi bittivirraksi. Pakkaamattomia formaatteja ovat esimerkiksi WAV-, AIFF- sekä AU-formaatit, kun taas pakattuja formaatteja ovat muun muassa MP3, FLAC, WMA, Vorbis ja AAC. Näytteen laataminen ja purkaminen näkyvät toiston aloitusviiveenä. Useimmiten kirjastot kuitenkin osaavat jättää ladatun näyteen laitteen muistiin, jolloin se on myöhemmillä toistokerroilla nopeammin toistettavissa ja niinpä aloitusviivekin on huomattavasti pienempi. Mittauksissa aloitusviive on huomioitu jättämällä ensimmäinen toisto mittaamatta.

Täytyy muistaa, että edellä mainitut formaatit kuten WAV ja AIFF ovat vain säiliömuotoja, joiden pääasiallinen tarkoitus on kuvailla sisältämänsä audiodatan koodaus. Useimmiten tiedostojen sisältämä ja audiokirjastojen sisäisesti käyttämä data on pulssikoodimoduloidussa muodossa. Kuten luvussa 2 todettiin, latenssiin jossain määrin vaikuttavat audion äänenlaadulliset ominaisuudet, kuten näytteenottotaajuus, bittisyvyys ja käytössä olevien kanavien määrä. Onkin siis hyvä mitata latensseja erilaisilla kombinaatioilla. Oletettavaa kuitenkin on, että formaatista riippumatta etenkin korkean tason kirjastot konvertoivat datan omaan sisäisesti käyttämään muotoonsa, jolloin sisäänmenevän audion laadullisilla ominaisuuksilla ei niinkään ole merkitystä.

Hyvä valinta audioformaatiksi mittauksiin on Wave-formaatti, sillä valitut lait-

teet ja kirjastot tukevat sitä poikkeuksetta. Sen sijaan tuetut näytteenottotaajuudet ja bittisyvyydet vaihtelevat jonkin verran kirjastosta ja alustasta riippuen. Esimerkiksi näytteenottotaajuudeltaan 44100 Hz ja bittisyvyydeltään 16-bittinen koodaus on lähes kaikkien kirjastojen tukema yhdistelmä. Mahdollisuuksien mukaan on syytä mitata kattavasti myös muilla arvoilla. Mittauksia varten valittiin seuraavat yhdistelmät:

- 8 kHz 8-bittinen mono (tai muu minimi arvo)
- 8 kHz 16-bittinen stereo
- 22,05 kHz 16-bittinen stereo
- 44,1 kHz 16-bittinen mono
- 44,1 kHz 16-bittinen stereo
- 48 kHz 16-bittinen stereo
- 96 kHz 16-bittinen stereo (tai muu maksimi arvo)
- 44,1 kHz 32-bittinen stereo
- 48 kHz 32-bittinen stereo
- 96 kHz 32-bittinen stereo (tai muu maksimi arvo)

Tutkimuksen kannalta ongelmaksi nouseekin konfiguraatioiden määrästä seuraava kombinatorinen räjähdys. Testattavia laitteita on 12 ja jokaista laitetta kohden täytyy mitata yhdestä neljään toteutusta eri kirjastoilla käyttäen kymmentä eri formaattia. Toistoja täytyy tehdä tarpeeksi monta, jotta voidaan määrittää edes jokseenkin tarkka keskiarvo ja laskea keskihajonta. Tekemällä jokaista kombinaatiota kohden kymmenen mittausta ja olettamalla, että tutkittavia kirjastototeutuksia on 2,5 per laite, yksittäisiä mittauksia tulisi noin kolme tuhatta. Käytännössä kaikilla eri formaateilla ei ole kuitenkaan ole mahdollista tai edes tarpeen tehdä mittauksia, joten lopullinen luku jäänee melko kauas edellä mainitusta arviosta.

# 7. AUDIOLATENSSIT JA HAVAINNOT

Tässä luvussa esitellään mittaustulokset ja analysoidaan alustakohtaisesti eri kirjastojen soveltuvuutta alhaista audiolatenssia vaativiin ohjelmistoihin. Samalla käydään läpi audio-ohjelmoinnin kannalta oleellisia kohtia mittauksia varten tuotetuista sovelluksista ja esitellään tiivistetyin esimerkein miten kirjastoja käytetään kooditasolla.

## 7.1 Mittaustulokset

Taulukkoon 7.1 on kerätty mittausten pohjalta laskettuja tunnuslukuja. Kaiken kaikkiaan onnistuneita mittauksia tehtiin yhteensä 2684 kappaletta. Taulukossa esiintyvien arvojen mittayksikkö on millisekunti. Jokaista laitteesta, audiokirjastosta ja -formaatista koostuvaa kombinaatiota kohti suoritettiin kymmenen mittausta. Näille kymmenen otoksen sarjoille laskettiin kesikarvo ja keskihajonta. Minimilatenssisarakkeessa ensimmäinen arvo on koko otoskannan pienin saavutettu yksittäinen latenssi. Jälkimmäinen arvo on pienin eri formaatteja kokeilemalla saatu keskiarvo, jonka mukaan tulokset on myös järjestetty. Minimihajontasarakkeessa on vastaavasti pienin saavutettu keskihajonta. Keskilatenssi- ja keskihajontasarakkeissa on kaikkien tietyllä laitteella ja kirjastolla mitattujen otosten pohjalta lasketut keskiarvo ja keskihajonta.

Latenssimittausten mukaan selkeästi parhaan tuloksen antaa Nokian Lumia 800 käytettäessä Silverlightin ja XNA:n yhdistelmää. Lumian tarjoama latenssi on noin 35 millisekunnin luokkaa, mikä on erittäin hyvä tulos. Lisäksi keskihajonta on varsin pieni. Hyvin suoriutuu myös iPhone käytettäessä RemoteIO-rajapintaa. Sen käyttö on kuitenkin merkittävästi XNA:n audiokirjastoja hankalampaa. Myös N9 pärjää kohtalaisen hyvin. Hieman yllättäenkin Nokian Series 40 -alusta suoriutuu melko tasavertaisesti iOS:n ja Harmattanin kanssa, joskin muilta ominaisuuksiltaan Series 40:n audiorajapinta on jokseenkin rajoittunut.

Kuten taulukosta 7.1 näkyy muut Windows Phone -alustaa käyttävät laitteet jäävät jonkin verran Lumiasta, mutta tulokset ovat kuitenkin tasaisen hyviä. BlackBerryn kohdalla latenssi alkaa olla jo kohtalaisen iso ja iOS:n korkeamman tason rajapintojen latenssi nousee jo 100 millisekunnin tietämille. iOS:n tulos on melkoisesti odotettua huonompi ottaen huomioon audioon tukeutuvien ohjelmien määrän App Storessa. Android- ja Symbian-alustat jäävät auttamatta muiden jalkoihin.

Mittauksissa ei päästy alle 100 ms latenssiin, joka on yksinkertaisesti liian häiritsevä musiikkisovelluksiin.

**Taulukko 7.1:** Latenssit ja hajonnat järjestettyinä minimilatenssin mukaan

| Laite | Alusta | Kirjasto | Minimilatenssi | Minimihajonta | Keskilatenssi | Keskihajonta |
|---|---|---|---|---|---|---|
| Nokia Lumia 800 | WP7.5 Mango | SL + XNA | 32,2 \| 34,7 | 1,3 | 37,4 | 3,4 |
| Nokia Lumia 800 | WP7.5 Mango | SL + XNA [1] | 32,5 \| 35,6 | 1,6 | 36,8 | 2,9 |
| Nokia N9 | Harmattan | QtMultimediaKit | 33,4 \| 39,8 | 3,0 | 44,4 | 5,5 |
| Nokia X3-02 | Series 40 6th Ed. | MMAPI | 32,6 \| 40,2 | 5,2 | 45,8 | 9,2 |
| Apple iPhone 4S | iOS 5 | RemoteIO | 35,6 \| 42,4 | 3,9 | 60,7 | 5,6 |
| Nokia Lumia 800 | WP7.5 Mango | XNA | 32,0 \| 47,8 | 7,7 | 52,4 | 9,4 |
| Samsung Omnia 7 | WP7.5 Mango | SL + XNA [1] | 49,7 \| 52,9 | 2,5 | 55,1 | 3,0 |
| HTC HD7 | WP7.5 Mango | SL + XNA | 46,8 \| 54,9 | 4,0 | 61,4 | 10,3 |
| Samsung Omnia 7 | WP7.5 Mango | SL + XNA | 51,9 \| 55,3 | 2,3 | 57,9 | 3,5 |
| Nokia N9 | Harmattan | PulseAudio | 34,0 \| 56,8 | 3,1 | 60,8 | 5,7 |
| HTC HD7 | WP7.5 Mango | XNA | 40,8 \| 60,9 | 7,7 | 65,8 | 9,4 |
| HTC HD7 | WP7 | SL + XNA | 45,4 \| 61,7 | 8,9 | 75,1 | 14,3 |
| Samsung Omnia 7 | WP7.5 Mango | XNA | 50,8 \| 65,6 | 6,5 | 69,6 | 9,1 |
| HTC HD7 | WP7 | XNA | 49,0 \| 72,4 | 12,8 | 82,2 | 15,2 |
| ZTE Blade | Android 2.3 | AudioTrack | 56,5 \| 73,0 | 8,6 | 79,0 | 9,6 |
| BB Torch 9800 | BlackBerry 6.0 | MMAPI | 67,1 \| 77,4 | 3,8 | 80,6 | 6,0 |
| iPhone 4S | iOS 5 | AudioToolbox | 75,6 \| 88,3 | 6,1 | 96,9 | 8,4 |
| ZTE Blade | Android 2.3 | OpenSL | 62,8 \| 93,6 | 8,3 | 95,2 | 10,4 |
| ZTE Blade | Android 2.3 | SoundPool | 88,5 \| 99,8 | 6,6 | 105,4 | 9,0 |
| ZTE Blade | Android 2.3 | MediaPlayer | 87,6 \| 100,9 | 8,0 | 109,5 | 11,2 |
| Apple iPhone 4S | iOS 5 | AVFoundation | 93,8 \| 107,4 | 6,1 | 112,8 | 8,4 |
| Samsung Nexus S | Android 2.3 | AudioTrack | 52,8 \| 111,8 | 24,7 | 142,2 | 39,2 |
| Nokia 701 | Symbian[3] Belle | DevSound | 101,0 \| 114,4 | 6,2 | 130,5 | 8,2 |
| Nokia N8 | Symbian[3] Anna | DevSound | 112,5 \| 128,7 | 7,4 | 137,1 | 13,4 |
| Nokia XM 5800 | Symbian[1] | QtMultimediaKit | 132,2 \| 136,0 | 0,5 | 139,6 | 5,2 |
| Samsung Nexus S | Android 2.3 | SoundPool | 135,0 \| 147,6 | 5,6 | 153,5 | 8,7 |
| Samsung Nexus S | Android 2.3 | MediaPlayer | 132,4 \| 149,0 | 7,9 | 150,9 | 8,6 |
| Samsung Nexus S | Android 2.3 | OpenSL | 103,4 \| 155,8 | 22,9 | 158,0 | 31,1 |
| Nokia 701 | Symbian[3] Belle | QtMultimediaKit | 157,8\| 178,6 | 7,9 | 193,5 | 11,1 |
| Nokia N8 | Symbian[3] Anna | MMF | 168,3 \| 195,9 | 6,7 | 205,1 | 13,5 |
| Nokia N8 | Symbian[3] Anna | QtMultimediaKit | 181,4 \| 198,6 | 8,2 | 208,4 | 13,4 |
| Nokia XM 5800 | Symbian[1] | DevSound | 315,3 \| 320,9 | 15,2 | 325,3 | 21,7 |

Taulukkoon 7.2 on kerätty mittaustulokset Drumkit-esimerkkisovellusta käyttäen. Jos keskilatensseja verrataan taulukon 7.1 vastaaviin arvoihin, nähdään että tulokset ovat jonkin verran heikompia, kuin mittauksia varten toteutetulla testisovelluksella. Tämä on toki odotettavaakin, koska Drumkiteissä on audiontoiston lisäksi paljon muutakin suoritinta kuormittavaa toiminnallisuutta. DevSound-toteutusten latensseista näkyy, että puskurin kokoon on täytynyt jättää hieman varmuusvaraa sen sijaan, että se olisi optimoitu vain parhaiten suoriutuvaa laitetta varten. Lisäksi voidaan todeta, että Windows Phone -alustalla on vielä jonkin verran parantamisen varaan. Samoin Harmattanilla voitaisiin siirtyä käyttämään PulseAudion sijaan QtMultimediaKit-rajapintoja.

**Taulukko 7.2:** Drumkit-esimerkkisovelluksella mitattuja latensseja

| Laite | Alusta | Kirjasto | Yksittäinen minimilatenssi | Keskilatenssi | Keskihajonta |
|---|---|---|---|---|---|
| Nokia X3-02 | Series 40 6th Ed. | MMAPI | 31,9 | 44,6 | 10,5 |
| Nokia Lumia 800 | WP7.5 Mango | XNA | 38,6 | 45,7 | 8,0 |
| Nokia N9 | Harmattan | PulseAudio | 56,8 | 62,2 | 3,9 |
| Samsung Omnia 7 | WP7.5 Mango | XNA | 53,6 | 67,7 | 8,5 |
| Nokia N8 | Symbian[3] Anna | DevSound | 121,2 | 133,8 | 8,4 |
| Nokia XM 5800 | Symbian[1] | MMAPI | 121,2 | 140,6 | 23,7 |
| Nokia 701 | Symbian[3] Belle | DevSound | 137,3 | 148,9 | 8,9 |
| Nokia XM 5800 | Symbian[1] | DevSound | 318,2 | 359,3 | 30,4 |

Sony Ericssonin Satio -puhelin päätettiin jättää vertailusta pois, koska laitteen

[1] Suoratoistototeutus

Qt-kirjastoja ei onnistuttu kohtuullisella vaivalla päivittämään, eikä toisaalta dedikoitujen versioiden tekeminen audiolatenssisovelluksista vaikuttanut houkuttelevalta vaihtoehdolta. Toinen testistä ulos jäänyt laite oli BlackBerry 7.0 -alustaa käyttävä Torch 9850, joka olisi edustanut 1,2 gigahertsin suorittimellaan Research In Motionin tehokkaampaa älypuhelinsukupolvea siinä missä nyt testattu laite Torch 9800 asettuu suoritinteholtaan Nokian N8:n ja ZTE Bladen kanssa samalle viivalle. Laitetta ei kuitenkaan onnistuttu hankkimaan mittauksia varten.

**Taulukko 7.3:** Sovelluskaupoissa jaeltavien musiikkisovellusten latensseja

| Laite | Alusta | Sovellus | Keskilatenssi | Keskihajonta |
|---|---|---|---|---|
| Nokia Lumia 800 | WP7.5 Mango | Cowbell | 80,5 | 12,2 |
| Nokia Lumia 800 | WP7.5 Mango | DrumKit | 144,0 | 22,6 |
| Nokia Lumia 800 | WP7.5 Mango | Drumkit | 251,9 | 7,7 |
| Nokia Lumia 800 | WP7.5 Mango | PianoPhone7 | 174,1 | 14,3 |
| Nokia Lumia 800 | WP7.5 Mango | PianistFree | 168,2 | 15,3 |
| Apple iPhone 4S | iOS 5 | Cowbell | 73,9 | 9,8 |
| Apple iPhone 4S | iOS 5 | GarageBand Drums | 52,5 | 5,2 |
| Apple iPhone 4S | iOS 5 | 3D Drumkit | 65,9 | 8,6 |
| Apple iPhone 4S | iOS 5 | Music Sparkle | 110,2 | 6,5 |
| Nokia N9 | Harmattan | Drumkit | 54,4 | 7,4 |
| Nokia N9 | Harmattan | VMPK & FluidSynth | 160,2 | 31,0 |
| ZTE Blade | Android 2.3 | Cowbell Hero | 507,8 | 67,0 |
| ZTE Blade | Android 2.3 | Drumkit | 134,9 | 10,9 |
| ZTE Blade | Android 2.3 | Drum Set Pro | 119,7 | 21,7 |

Vertailun vuoksi testissä parhaiten pärjänneille laitteille asennettiin sovelluskaupoista saatavilla olevia soittosovelluksia ja mitattiin myös niiden latenssit. Suurin osa ladatuista sovelluksista oli ilmaisia, joten kovin korkeaa tasoa ei kai voinutkaan odottaa. Taulukon 7.3 tuloksista kuitenkin näkyy, että aina latenssiin ei ole kiinnitetty huomiota. Esimerkiksi Nokian Lumialla pitäisi helposti päästä huipputuloksiin, mutta esimerkiksi Drumkit-sovelluksen kohdalla latenssi on onnistuttu kasvattamaan jopa sekunnin neljännekseen. Androidilla Cowbell Hero oli täysin luokaton esitys ja latenssi onkin laskettu vain kolmen onnistuneen mittauksen perusteella. Muut mittausyritykset epäonnistuivat, koska lehmänkellon sointi oli niin summittaista, ettei sitä voinut yhdistää näytön napautukseen. Pianosovellusten kohdalla korkeita latensseja saattaisi selittää mahdollinen MIDI-rajapintojen käyttö. Toki myös Drumkit-sovellusten yhteydessä on mahdollista, että rumpuäänien tuottamiseen on käytetty MIDI-standardin kanavaa 10, joka on varattu perkussioäänille.

Parhaaseen tulokseen pääsee GarageBand, mutta se ei tosin ole mikään yllätys, koska sovellus on maksullinen ja Applen tuottama. Toisaalta vastaavaan latenssiin pääsee myös Harmattan alustalle suunnattu ilmainen Drumkit-sovellus. Myös 3D Drumkit yltää kohtuulliseen latenssiin. Selvästikään kolmiulotteisuudella ei juurikaan ole vaikutusta audion toiston suorituskykyyn.

### 7.1.1 BlackBerry

BlackBerryllä mitattiin Java ME:n Mobile Media -rajapintaa käyttävä toteutus kuten Series 40:lläkin. Rajapinta on periaatteessa hyvin helppokäyttöinen. Kooditasolla äänileikkeen toistaminen tapahtuu listauksen 7.1 ja 7.2 mukaisesti. Ensiksi luo-

daan siis uusi Player-instanssi ja alustetaan se sound.wav-nimisellä äänileikkeellä. Lisäksi kutsutaan prefetch-metodia, joka lataa äänileikkeen muistiin, jotta toistovaiheessa ei enää aiheudu viivettä äänileikkeen lataamisesta. Koodilistauksessa 7.2 asetetaan toistokohta leikkeen alkuun ja toistetaan äänileike.

**Listaus 7.1:** Player-instanssin luominen Mobile Media API:a käyttäen

```
1 try {
2     InputStream is = this.getClass().getResourceAsStream("/sound.wav");
3     Player player = Manager.createPlayer(is, "audio/wav");
4     player.prefetch();
5 catch (IOException ioe) {...}
6 catch (MediaException me) {...}
```

**Listaus 7.2:** Äänileikkeen toistaminen Mobile Media API:n avulla

```
1 try {
2     player.setMediaTime(0);
3     player.start();
4 } catch (MediaException me) {...}
```

Mittaustulos ei ole mitenkään erityisen mairitteleva vaan pikemminkin keskinkertainen. BlackBerry sijoittuukin noin puoliväliin tulostaulukkoa vajaan 80 millisekunnin latenssillaan. Tulos on merkittävästi Nokian X3-02-puhelimen latenssia heikompi, eikä oikein sovellu musiikkisovellusten toteuttamiseen. Selvää onkin, että vaikka eri laitteet tarjoaisivat saman rajapinnan, voivat toteutukset poiketa toisistaan hyvinkin paljon. Yhdessä laitteistoerojen kanssa tulokset eivät välttämättä ole millään tavalla yhteydessä toisiinsa.

### 7.1.2 Series 40

Series 40 pärjäsi latenssimittauksissa varsin hyvin. Osasyy hyvään tulokseen lienee itse alustan keveys ja yhtäaikaisten prosessien vähyys. Series 40:ssä ei ole moniajoa, joten raskaiden taustaprosessien tuomaan prosessorikuormaan ei tarvitse varautua. Näin ollen puskurikokokin voidaan pitää pienenä. Hieman ristiriitainen tulos on kuitenkin kohtalaisen iso keskihajonta, sillä yhtäaikaisten prosessien vähyyden voisi kuvitella johtavan tasaisempaan audiolatenssin.

Audioparametrien säätämiseen ei juurikaan ole mahdollisuuksia. Esimerkiksi puskurin kokoon, eikä ulosmenevän audion näytteeottotaajuuteen voi vaikuttaa. Niinpä keinot latenssiin vaikuttamiseksi ovat vähissä. Onneksi siihen ei oikeastaan ole tarvettakaan.

Latenssin kannalta erityisen mielenkiintoinen havainto oli, että toistettaessa kerrallaan vain yhtä äänileikettä latenssi on melko korkea, mutta heti kun ohjelmaan

ladataan toinenkin äänileike, latenssi putoaa merkittävästi. Ilmiöllä lienee jotain yhteyttä miksaamisen kanssa. Mahdollisesti miksausalgoritmia käytettäessä – syystä tai toisesta – käytetään pienempää puskuria ulosmenevälle audiolle.

Rajapinnan rajoitteet aiheuttavat hieman ylimääräistä pään vaivaa, jos halutaan toistaa useita eri äänileikkeitä yhtäaikaa. Series 40 rajoittaa yhtäaikaisten Player-instanssien määrän kahdeksaan. Lisäksi muistirajoitteista johtuen vain osa Player-instansseista voi soida yhtäaikaa. Jos liian montaa äänileikettä yritetään soittaa yhtäaikaisesti, heittää start-metodi poikkeuksen. Niinpä esimerkeksi peleissä, joissa käytetään useita ääniefektejä, on tilannetta hyvä monitoroida esimerkiksi siten, että toiston epäonnistuessa pysäytetään kauimmin soinut ääni ja yritetään toistoa uudestaan. Taustamusiikille on tietenkin hyvä varata oma Player-instanssinsa.

### 7.1.3 Symbian

Symbianille toteutettiin QtMultimediaKit-, Multi Media Framework Client API - sekä DevSound-pohjaiset testisovellukset. Koska Qt GameEnabler käyttää audion toistamiseen QtMultimediaKitin tapaan QAudioOutput-luokkaa, Qt GameEnableria käytettiin lopulta ainoastaan DevSoundin ja PulseAudion yhteydessä helpottamaan datan lataamisesta ja käyttämistä. Qt GameEnabler sisältää myös QAudiolle vaihtoehtoisen toteutuksen, mutta sen kokeellisen luonteen vuoksi, sitä ei sisällytetty mittauksiin.

MMF-kirjastoa käyttäen toteutettiin sekä MdaAudioSamplePlayer-rajapintaa käyttävä toteutus, että MdaAudioOutputStream-rajapintaa hyödyntävä suoratoistollinen toteutus. Multi Media Framework tukeutuu kuitenkin DevSoundiin, joten lähtökohtaisestikaan ei ollut todennäköistä, että sillä voitaisiin saavuttaa DevSoundia parempaa latenssia. Tästä syystä MMF-mittauksia tyydyttiinkin suorittamaan vain yhdellä laitteella, että voitiin todeta DevSound-toteutuksen paremmuus.

Kaiken kaikkiaan Symbianin audion toisto ei ole kovin vakuuttavaa. Latenssi on keskimäärin joukon heikointa, eikä alusta siis oikein sovellu latenssikriittisten sovellusten toteuttamiseen. Pienimmän latenssin tarjoilee DevSound, mikä ei ole sinänsä yllättävää, koska se sijoittuu lähimmäs laitteistoa, ja lopulta Symbianin kaikki audio toteutukset toimivat DevSoundin päällä.

Oletuksena puskurin koko on kiinteä 4096 tavua, josta aiheutuu hyvin huomattava viive. On kuitenkin mahdollista täyttää puskuri vain osittain, jolloin se päätyy toistettavaksi nopeammin ja latenssi voidaan pudottaa murto-osaan. Tällöin kuitenkin täytyy olla tarkkana, ettei dataa syötetä liian pienissä erissä, sillä se aiheuttaisi ääneen katkonaisuutta. Koska Symbian tukee moniajoa, prosessorin käyttö voi vaihdella suuresti. Tämä kannattaa huomioida puskurin kokoa valittaessa. Listauksessa 7.3 on lyhennelty esimerkki DevSoundin käytöstä.

**Listaus 7.3:** Äänileikkeen toistaminen DevSound-rajapinnan avulla

```
#include <mmf/server/sounddevice.h>

class DevSoundPlayer : public MDevSoundObserver {
  public:
    DevSoundPlayer() {
      devSound = CMMFDevSound::NewL();
      devSound->InitializeL(*this, KMMFFourCCCodePCM16, EMMFStatePlaying);
    }

  private:
    CMMFDevSound* devSound; // DevSound-instanssi
    CMMFDescriptorBuffer* audio; // Audiodata

    void InitializeComplete(TInt aError) {
      TMMFCapabilities caps = m_devSound->Config();
      caps.iChannels = EMMFStereo;
      caps.iEncoding = EMMFSoundEncoding16BitPCM;
      caps.iRate = EMMFSampleRate44100Hz;

      devSound->SetConfigL(caps);
      devSound->SetVolume(devSound->MaxVolume());
      devSound->PlayInitL(); // Toistetaan audio
    }

    void LoadAudio(TFileName fileName) {
    ...
```

Lisäksi täytyy toteuttaa takaisinkutsu-funktio, joka täyttää puskurin audiodatalla. Listauksessa 7.4 on esimerkki takaisinkutsufunktion toteutuksesta.

QtMultimediaKit-moduulia käytettiin QML:stä käsin (koodiesimerkki Harmattanin yhteydessä). Mielenkiintoista on, että MultimediaKit-toteutus Nokian XpressMusic 5800:lla pääsee melko lähelle N8:n DevSound-toteutuksen latenssia, kun taas N8:lla MultimediaKit-toteutus tuottaa huomattavasti heikomman tuloksen. Ero saattaa selittyä äänipiirien ominaisuuksilla. Hämmentävää on kuitenkin se, että XpressMusicilla DevSound-toteutuksen toisto oli erittäin hidas. Näyttäisi siltä, että puskurin osittaisella täyttämisellä ei ole vastaavaa vaikutusta kuin esimerkiksi N8:lla.

**Listaus 7.4:** DevSound-takaisinkutsu

```
1     void BufferToBeFilled(CMMFBuffer* aBuffer) {
2       CMMFDataBuffer* buffer = static_cast<CMMFDataBuffer*>(aBuffer);
3
4       // Puskurin oletuskoko on 4096 tavua. Pienennetään latenssia
5       // täyttämällä puskuri vain osaksi
6       //TInt requestSize = buffer->RequestSize();
7       TInt requestSize = 1024;
8
9       if(audio->Data().Length() < requestSize) {
10        buffer->SetLastBuffer(ETrue);
11      }
12
13      buffer->Data().Copy(audio->Data().LeftTPtr(requestSize));
14      audio->Data().Delete(0, requestSize);
15      if(buffer->Data().Length() > 0) {
16        devSound->PlayData();
17      }
18    }
19 };
```

### 7.1.4 Harmattan

Harmattan on ristiriitainen alusta. Kuten audiotoistoltaan myös muilta osa-alueiltaan Harmattan on tällä hetkellä ehdotonta mobiilikäyttöjärjestelmien eliittiä. Kuitenkaan Harmattanin tulevaisuus ei näytä kovin valoisalta, vaikka potentiaalia alustassa olisi hurjasti. Mittaustuloksissa Harmattan sijoittuu lähelle terävintä kärkeä. Lisäksi Multimedia QML Pluginin ansiosta audiota on myös äärimmäisen helppo käyttää, kuten listaus 7.5 osoittaa.

**Listaus 7.5:** Äänileikkeen toistaminen QML:ssä

```
1  SoundEffect {
2          id: sound
3          source: "sound.wav"
4  }
5
6  MouseArea {
7          anchors.fill: parent
8          onPressed: sound.play()
9  }
```

Toisaalta PulseAudio-kirjasto, tarjoaa monipuoliset aseet audion toistoon, jos audiodataa täytyy vaikkapa käsitellä lennossa ennen toistamista. Kirjaston käyttö on toki merkittävästi monimutkaisempaa ja audiolatenssiin edes suoratoistolla ei enää saavuteta merkittäviä parannuksia.

### 7.1.5 iOS

Applen iOS on tunnetusti kykenevä alhaiseen latenssiin, mutta sen saavuttaminen ei ole ihan triviaalia. Korkean tason kirjastot AVFoundation ja AudioToolbox ovat kumpikin hyvin helppokäyttöisiä, mutta niiden tarjoama latenssi on yllättävänkin heikko. RemoteIO Audio Unitia käytettäessä moni kertoo päässeensä 10 ms:n latenssiin, mutta mittauksissa latenssi oli pienimillään noin 40 ms:n luokkaa. Toki mukana on myös kosketusviive, joten tuloksia ei voitane verrata. Listauksissa 7.6 ja 7.7 nähdään esimerkit AVFoundation- ja AudioToolbox-toteutuksista.

**Listaus 7.6:** Äänileikkeen toistaminen AVFoundation-kirjaston avulla

```
1  // AVFoundation
2  #import <AVFoundation/AVAudioPlayer.h>
3
4  AVAudioPlayer *audioPlayer;
5
6  NSURL *fileURL = [[NSBundle mainBundle] URLForResource:@"sound"
7    withExtension:@"wav"];
8  AVAudioPlayer *audioPlayer = [[AVAudioPlayer alloc]
9    initWithContentsOfURL:fileURL error:nil];
10 [audioPlayer prepareToPlay];
11 audioPlayer.currentTime = 0;
12 [audioPlayer play];
```

**Listaus 7.7:** Äänileikkeen toistaminen AudiotToolbox-kirjaston avulla

```
1  // AudioToolbox
2  #import <AudioToolbox/AudioToolbox.h>
3
4  NSURL *fileURL = [[NSBundle mainBundle] URLForResource:@"sound"
5    withExtension:@"wav"];
6  SystemSoundID audioSSID = 0;
7  AudioServicesCreateSystemSoundID((__bridge CFURLRef) fileURL, &audioSSID);
8
9  AudioServicesPlaySystemSound(audioSSID);
```

Kuten huomataan, äänileikkeen toistaminen vaatii vain muutamia koodirivejä. Toisaalta monipuolisemman audiokontrollin saavuttamiseksi vaaditaan moninkertainen määrä koodia. Seuraavissa listauksissa käydään läpi riisuttu esimerkki audion toistamisesta RemoteIO:n avulla. Listauksessa 7.8 määritellään ensiksi audioformaatti ja alustetaan sen jälkeen audiosessio. Tilan säästämiseksi audiotiedosto on esimerkissä vain epämääräinen synthesize-määrellä varustettu viite.

**Listaus 7.8:** Audioformaatin ja audiosession alustaminen

```
#import "AudioUnit/AudioUnit.h"
#import "AudioToolbox/AudioToolbox.h"

@synthesize audioFile; // Audiodata
AudioComponentInstance audioUnit;

-(void)play { AudioOutputUnitStart(audioUnit); }
-(void)stop { AudioOutputUnitStop(audioUnit); }

-(void)init {
  // Määritellään audioformaatti
  AudioStreamBasicDescription audioFormat;
  audioFormat.mSampleRate   = 44100.0f;
  audioFormat.mFormatID   = kAudioFormatLinearPCM;
  audioFormat.mFormatFlags  = kAudioFormatFlagIsSignedInteger |
    kAudioFormatFlagIsPacked;
  audioFormat.mFramesPerPacket = 1;
  audioFormat.mChannelsPerFrame = 2;
  audioFormat.mBitsPerChannel  = 16;
  audioFormat.mBytesPerPacket  = 4;
  audioFormat.mBytesPerFrame  = 4;

  AudioUnitSetProperty(audioUnit, kAudioUnitProperty_StreamFormat,
    kAudioUnitScope_Input, 0, &audioFormat, sizeof(audioFormat));

  // Alustetaan audiosessio
  AudioSessionInitialize(NULL, NULL, NULL, NULL);
  UInt32 audioCategory = kAudioSessionCategory_MediaPlayback;
  AudioSessionSetProperty(kAudioSessionProperty_AudioCategory,
    sizeof(audioCategory), &audioCategory);
  ...
```

Listauksen 7.8 alussa määritellään käytettävä puskuri. PreferredBufferSize on huomionarvoinen ominaisuus, jolla voidaan pyrkiä vaikuttamaan audiolatenssiin.

Mittausten perusteella määreen asettamisesta olikin hyötyä, sillä pienin latenssi saavutettiin sen avulla. Simulaattoria käytettäessä vaikutusta ei huomaa. Kun muut alkuvalmistelut on saatu tehtyä, otetaan RemoteIO-audioyksikkö käyttöön ja asetetaan sille takaisinkutsufunktio puskurin täydentämiseksi.

**Listaus 7.9:** Audiopuskurin määrittäminen

```
1   ...
2   // Asetetaan toivottu puskurin koko sekunneissa
3   Float32 preferredBufferSize = .002;
4   AudioSessionSetProperty(
5     kAudioSessionProperty_PreferredHardwareIOBufferDuration,
6     sizeof(preferredBufferSize), &preferredBufferSize);
7
8   // Pyydetään puskurin koko
9   Float32 audioBufferSize;
10  UInt32 size = sizeof (audioBufferSize);
11  AudioSessionGetProperty(
12    kAudioSessionProperty_CurrentHardwareIOBufferDuration,
13    &size, &audioBufferSize);
14  AudioSessionSetActive(true);
15
16  // Otetaan RemoteIO-audioyksikkö käyttöön
17  AudioComponentDescription desc;
18  desc.componentType = kAudioUnitType_Output;
19  desc.componentSubType = kAudioUnitSubType_RemoteIO;
20  desc.componentFlags = 0;
21  desc.componentFlagsMask = 0;
22  desc.componentManufacturer = kAudioUnitManufacturer_Apple;
23  AudioComponentInstanceNew(AudioComponentFindNext(NULL, &desc), &audioUnit);
24  UInt32 flag = 1;
25  AudioUnitSetProperty(audioUnit, kAudioOutputUnitProperty_EnableIO,
26    kAudioUnitScope_Output, 0, &flag, sizeof(flag));
27
28  // Asetetaan takaisinkutsu ja mukaan referenssi audioPlayerin
29  AURenderCallbackStruct callbackStruct;
30  callbackStruct.inputProc = playCallback;
31  callbackStruct.inputProcRefCon = (__bridge void *) self;
32  AudioUnitSetProperty(audioUnit, kAudioUnitProperty_SetRenderCallback,
33    kAudioUnitScope_Global, 0, &callbackStruct, sizeof(callbackStruct));
34
35  AudioUnitInitialize(audioUnit);
36 }
```

Listauksessa 7.8 asetettu takaisinkutsufunktio tätyy tottakai myös toteuttaa. Niinpä listauksessa 7.10 on vielä esimerkki playCallback-funktion toteutuksesta, joka olettaa, että audioFile-luokalla on getNextFrame-metodi audiodatan lukemiseksi, sekä EndOfFile-lippu, josta voidaan päätellä onko tiedoston loppu jo saavutettu.

**Listaus 7.10:** RemoteIO:n suoratoistossa tarvittava takaisinkutsufunktio

```
1  static OSStatus playCallback(void *inRefCon, AudioUnitRenderActionFlags
2    *ioActionFlags, const AudioTimeStamp *inTimeStamp, UInt32 inBusNumber,
3    UInt32 inNumberFrames, AudioBufferList *ioData) {
4      @autoreleasepool {
5        AudioPlayer *audioPlayer = (__bridge AudioPlayer *)inRefCon;
6
7        // Käydään puskurit läpi
8        for (int i = 0 ; i < ioData->mNumberBuffers; i++){
9          AudioBuffer buffer = ioData->mBuffers[i];
10         UInt32 *frameBuffer = buffer.mData;
11
12         // Täytetään puskuri
13         for (int j = 0; j < inNumberFrames; j++){
14           // Noudetaan seuraava audiokehys
15           frameBuffer[j] = [[audioPlayer audioFile] getNextFrame];
16           if([[audioPlayer audioFile] endOfFile] == YES) {
17             [audioPlayer stop];
18           }
19         }
20       }
21     }
22     return noErr;
23 }
```

### 7.1.6 Android

Audio ei ole Androidin ominta aluetta. Latenssi on luvattoman suuri jo pelkästään tavalliseen pelikäyttöön, saati sitten Drumkitin kaltaisiin sovelluksiin. Jokseenkin yllättävsti Samsungin Nexus S suoriutuu mittausten perusteella huomattavasti ZTE:n halpamallia huonommin. Sen lisäksi, että latenssi on kaikkien kirjastojen kohdalla suurempi, myös hajonta on AudioTrackin ja OpenSL:n kohdalla huomattavan iso.

Harmattanin tavoin myös Android toimii Linuxin päällä. Ihan yhtä vapaasti ei kuitenkaan natiivikirjastoja pysty Androidissa käyttämään. Versiosta 2.3 lähtien audion toistoon on ollut tarjolla OpenSL-kirjasto, jota voi käyttää Android NDK:n avulla. Sekään ei kuitenkaan tuo merkittäviä parannuksia latenssiin, mutta tarjoaa

muun muassa suoratoisto-ominaisuudet. Audiodatan suoratoisto on kuitenkin mahdollista myös käyttämällä AudioTrack-rajapintaa, joten onkin hieman hankala keksiä natiivikirjastolle käyttötapauksia. Toisaalta AudioTrackiin liittyen on raportoitu joitakin ongelmallisia bugeja, joten OpenSL voisi toimia hyvänä varasuunnitelmana.

Tulosten perusteella pienin latenssi on saavutettavissa käyttämällä AudioTrack-rajapintaa. Listauksessa 7.11 on näyte rajapinnan käytöstä kooditasolla.

**Listaus 7.11:** Äänileikkeen toistaminen AudioTrack-rajapinnan avulla

```
1  import android.media.AudioFormat;
2  import android.media.AudioManager;
3  import android.media.AudioTrack;
4  ...
5  short[] audio = sound(); // Generoidaan audiodataa
6
7  // Käytetään pienintä mahdollista puskuria
8  int minSize = AudioTrack.getMinBufferSize( 44100,
9    AudioFormat.CHANNEL_CONFIGURATION_STEREO,
10   AudioFormat.ENCODING_PCM_16BIT);
11
12 // Luodaan ääniraita
13 AudioTrack track = new AudioTrack( AudioManager.STREAM_MUSIC, 44100,
14   AudioFormat.CHANNEL_CONFIGURATION_STEREO,
15   AudioFormat.ENCODING_PCM_16BIT, minSize, AudioTrack.MODE_STREAM);
16 track.play();
17
18 // Kirjoitetaan näytteet ääniraitaan
19 track.write( audio, 0, audio.length );
```

Helppokäyttöisemmistä rajapinnoista SoundPool vaikuttaisi päihittävän MediaPlayerin, mutta käytännössä ero on suhteellisen pieni. MediaPlayer sopii esimerkiksi musiikin toistamiseen, kun taas SoundPool soveltuu paremmin lyhyiden äänileikkeiden toistamiseen etenkin, jos tarkoituksena on toistaa useampia leikkeitä yhtäaikaisesti kuten vaikkapa peleissä. Listauksessa 7.12 toistetaan äänleike käyttäen MediaPlayer-rajapintaa ja listauksessa 7.13 käytetään SoundPool-rajapintaa.

**Listaus 7.12:** Äänileikkeen toistaminen MediaPlayer-rajapinnan avulla

```
1  // MediaPlayer
2  import android.app.Activity;
3  import android.media.MediaPlayer;
4
5  MediaPlayer mp = MediaPlayer.create(getApplicationContext(), R.raw.sound);
6  mp.start();
```

**Listaus 7.13:** Äänileikkeen toistaminen SoundPool-rajapinnan avulla

```
1  // SoundPool
2  import android.media.AudioManager;
3  import android.media.SoundPool;
4
5  SoundPool soundPool = new SoundPool(1, AudioManager.STREAM_MUSIC, 0);
6  int soundId = soundPool.load(this, R.raw.sound, 1);
7  int streamId = soundPool.play(soundId, 1.0f, 1.0f, 0, 0, 1.0f);
```

OpenSL on huomattavasti edeltäviä esimerkkejä vaikeammin lähestyttävä. Rajapintaa käyttävät metodit kirjoitetaan C:llä. Jotta funktioita voitaisiin käyttää Android-sovelluksesta käsin, metodit sidotaan Javaan JNI-kehyksen avulla (Java Native Interface) ja C-toteutus käännetään Android NDK:n sisältämän ndk-build-työkalun avulla so-päätteisiksi tiedostoiksi. Kun so-tiedostot on käännetty, ne lisätään projektiin, minkä jälkeen niiden metodeita voidaan käyttää Java-koodissa listauksessa 7.14 esitetyllä tavalla. Listausten 7.15 ja 7.16 esittämä C-toteutus on muunnelma NDK:n NativeAudio.java-esimerkistä:

**Listaus 7.14:** JNI:n avulla sidottujen funktioiden kutsuminen Javasta käsin

```
1  public class AudioActivity extends Activity {
2    @Override
3    public void onCreate(Bundle savedInstanceState) {
4      super.onCreate(savedInstanceState);
5      setContentView(R.layout.main);
6      init();
7      play();
8    }
9
10   // Natiivi-metodit
11   public static native void init();
12   public static native void play();
13
14   // Ladataan jni-kirjasto
15   static {
16     System.loadLibrary("native-audio-jni");
17   }
18 }
```

**Listaus 7.15:** Audioformaatin määrittäminen OpenSL-natiivirajapinnassa

```
#include <jni.h>
#include <SLES/OpenSLES.h>
#include "SLES/OpenSLES_Android.h"
#include <sys/types.h>

static SLObjectItf engineObject = NULL;
static SLEngineItf engineEngine;
static SLObjectItf outputMixObject = NULL;
static SLObjectItf bqPlayerObject = NULL;
static SLPlayItf bqPlayerPlay;
static SLAndroidSimpleBufferQueueItf bqPlayerBufferQueue;

static const char sound[] =
#include "sound.h" // Audiodata
;

static short *buffer;
static unsigned bufferSize;

// Yksinkertaistettu takaisinkutsu puskurin täyttämiseksi
void bqPlayerCallback(SLAndroidSimpleBufferQueueItf bq, void *context) {
  (*bqPlayerBufferQueue)->Enqueue(bqPlayerBufferQueue, buffer, bufferSize);
}

void Java_audio_latency_AudioActivity_init(JNIEnv* env, jclass clazz) {
  // Luodaan engine
  slCreateEngine(&engineObject, 0, NULL, 0, NULL, NULL);
  (*engineObject)->Realize(engineObject, SL_BOOLEAN_FALSE);

  // Asetetaan käytettävä audioformaatti
  SLDataLocator_AndroidSimpleBufferQueue loc_bufq =
    {SL_DATALOCATOR_ANDROIDSIMPLEBUFFERQUEUE, 2};
  SLDataFormat_PCM format_pcm = {SL_DATAFORMAT_PCM, 1,
    SL_SAMPLINGRATE_44_1, SL_PCMSAMPLEFORMAT_FIXED_16,
    SL_PCMSAMPLEFORMAT_FIXED_16, SL_SPEAKER_FRONT_CENTER,
    SL_BYTEORDER_LITTLEENDIAN};
  SLDataSource audioSrc = {&loc_bufq, &format_pcm};
  ...
```

Kun audioformaatti on saatu määriteltyä ja ulostulo on konfiguroitu onnistuneesti, voidaan luoda soitinistanssi ja määrittää play-metodi toiston aloittamista varten.

Listauksessa 7.16 jatketaan suoraan siitä mihin listauksessa 7.15 jäätiin. Todellisuudessa audiodata tulisi toistaa pienissä osissa, mutta latenssimittauksiin riitti, että audiodata on yhden puskurin mittainen. Lisäksi esimerkistä on jätetty pois siivoustoimenpiteet, jotka löytyvät kyllä alkuperäisestä esimerkistä.

**Listaus 7.16:** Soittimen alustaminen ja äänileikkeen toistaminen OpenSL-natiivirajapinnan avulla

```
1   ...
2     // Konfiguroidaan ulostulo
3   SLDataLocator_OutputMix loc_outmix = {SL_DATALOCATOR_OUTPUTMIX,
4     outputMixObject};
5   SLDataSink audioSnk = {&loc_outmix, NULL};
6
7   // Luodaan soitin
8   const SLInterfaceID ids[2] = {SL_IID_BUFFERQUEUE, SL_IID_EFFECTSEND};
9   const SLboolean req[2] = {SL_BOOLEAN_TRUE, SL_BOOLEAN_TRUE};
10  (*engineEngine)->CreateAudioPlayer(engineEngine, &bqPlayerObject,
11    &audioSrc, &audioSnk, 2, ids, req);
12  (*bqPlayerObject)->Realize(bqPlayerObject, SL_BOOLEAN_FALSE);
13
14  // Pyydetään tarvittavat rajapinnat
15  (*bqPlayerObject)->GetInterface(bqPlayerObject, SL_IID_PLAY,
16    &bqPlayerPlay);
17  (*bqPlayerObject)->GetInterface(bqPlayerObject, SL_IID_BUFFERQUEUE,
18    &bqPlayerBufferQueue);
19
20  // Rekisteröidään takaisinkutsu
21  (*bqPlayerBufferQueue)->RegisterCallback(bqPlayerBufferQueue,
22    bqPlayerCallback, NULL);
23
24  // Aloitetaan toisto
25  (*bqPlayerPlay)->SetPlayState(bqPlayerPlay, SL_PLAYSTATE_PLAYING);
26
27  // Asetetaan audiodata puskuriin
28  buffer = (short *) sound;
29  bufferSize = sizeof(sound);
30 }
31
32 jboolean Java_audio_latency_AudioActivity_play(JNIEnv* env, jclass clazz) {
33  (*bqPlayerBufferQueue)->Enqueue(bqPlayerBufferQueue, buffer, bufferSize);
34 }
```

### 7.1.7 Windows Phone

Windows Phone pärjäsi mittauksissa kohtuullisesti. XNA-version tuloksista voidaan havaita suurehko keskihajonta. Hajonta johtuu pääasiassa kosketustapahtumien lukemisesta XNA:n pelisilmukassa, joka päivittyy enintään vain 30 kertaa sekunnissa. Tästä aiheuttuu jo maksimissaan n. 33 ms viive. Mango-päivityksen myötä päivitysnopeus on mahdollista nostaa maksimissaan 60 kertaan sekunnissa. Niinpä hajonta lähes puolittuu ja latenssin keskiarvo pienenee kohtuulliseksi. Listaus 7.17 näyttää miten XNA:n avulla voidaan toistaa äänileike.

**Listaus 7.17:** Kosketusinformaation lukeminen ja äänileikkeen toistaminen XNA:ssa

```
1  public class Main : Microsoft.Xna.Framework.Game {
2    SoundEffect sound;
3
4    public Main() {
5      // Päivitystiheys 60 fps
6      TargetElapsedTime = TimeSpan.FromTicks(166666);
7    }
8
9    protected override void LoadContent() {
10     sound = Content.Load<SoundEffect>("sound");
11   }
12
13   protected override void Update(GameTime gameTime) {
14     TouchCollection touches = TouchPanel.GetState();
15     foreach (TouchLocation touch in touches) {
16       if (touch.State == TouchLocationState.Pressed) {
17         sound.Play();
18         break;
19       }
20     }
21     base.Update(gameTime);
22   }
23 }
```

Silverlight-versiossa kosketustapahtumia ei ole sidottu pelisilmukkaan, vaan niihin voidaan reagoida asynkronisesti. Alustaversio 7.0 ilmeisesti kärsii kuitenkin ohjelmointivirheestä, sillä kosketustapahtuman tyyppi ei ole aina sitä mitä pitäisi. Mango-päivitys näyttäisi korjaavan ongelman, minkä ansiosta paras ja tasaisin latenssi saavutetaankin Silverlight-XNA-yhdistelmällä. Toteutuksesta nähdään esimerkki listauksessa 7.18.

**Listaus 7.18:** Kosketustapahtumiin reagointi Silverlight-sovelluksessa sekä äänileikkeen toistaminen XNA:ta hyödyntäen

```
1  public partial class MainPage : PhoneApplicationPage {
2    SoundEffect sound;
3
4    public MainPage() {
5      sound = SoundEffect.FromStream(TitleContainer.OpenStream("sound.wav"));
6
7      // Käynnistetään XNA:n päivityssilmukka
8      DispatcherTimer XnaDispatchTimer = new DispatcherTimer();
9      XnaDispatchTimer.Interval = TimeSpan.FromMilliseconds(50);
10     XnaDispatchTimer.Tick += delegate {
11       try {
12         FrameworkDispatcher.Update();
13       } catch { }
14     };
15     XnaDispatchTimer.Start();
16
17     // Rekisteröidytään kuuntelemaan matalan tason kosketustapahtumia
18     Touch.FrameReported += new TouchFrameEventHandler(Touch_FrameReported);
19   }
20
21   void Touch_FrameReported(object sender, TouchFrameEventArgs e) {
22     TouchPointCollection points = e.GetTouchPoints(this);
23     foreach (TouchPoint point in points) {
24       if (point.Action == TouchAction.Down) {
25         sound.Play();
26       }
27     }
28   }
29 }
```

Aikaisemmista esimerkeistä on tilan säästämiseksi jätetty kosketustapahtumiin reagointi pois, koska usein sen toteuttaminen on hyvin triviaalia. Windows Phone -ympäristössä XNA- ja Silverlight-toteutusten suurin eroavuus on juuri kosketustapahtumiin reagointi ja siitä syystä ne onkin sisällytetty esimerkkeihin. Kaiken kaikkiaan XNA:n audiorajapinta on erittäin mukava käyttää ja tarjoaa laitteesta riipuen vähintäänkin kohtuullisen lopputuloksen. Myös suoratoisto on rajapinnan avulla mahdollista, mutta latenssia parantaminen ei sen avulla enää onnistunut.

**Listaus 7.19:** Audiotiedoston lukeminen suoratoistoa varten

```
1  public partial class MainPage : PhoneApplicationPage {
2    DynamicSoundEffectInstance dynamicSound;
3    int position;
4    int bufferSize;
5    byte[] byteArray;
6
7    // Constructor
8    public MainPage() {
9      // Luetaan otsikkotiedot
10     Stream waveFileStream = TitleContainer.OpenStream("sound.wav");
11     BinaryReader reader = new BinaryReader(waveFileStream);
12     int chunkID = reader.ReadInt32();
13     int fileSize = reader.ReadInt32();
14     int riffType = reader.ReadInt32();
15     int fmtID = reader.ReadInt32();
16     int fmtSize = reader.ReadInt32();
17     int fmtCode = reader.ReadInt16();
18     int channels = reader.ReadInt16();
19     int sampleRate = reader.ReadInt32();
20     int fmtAvgBPS = reader.ReadInt32();
21     int fmtBlockAlign = reader.ReadInt16();
22     int bitDepth = reader.ReadInt16();
23
24     if (fmtSize == 18) {
25       int fmtExtraSize = reader.ReadInt16();
26       reader.ReadBytes(fmtExtraSize);
27     }
28
29     int dataID = reader.ReadInt32();
30     int dataSize = reader.ReadInt32();
31     byteArray = reader.ReadBytes(dataSize);
32     ...
```

Listauksessa 7.19 esitetään miten äänileikkeen audiodata luetaan tiedostosta. Ladattu äänileike voidaan toistaa DynamicSoundEffectInstance-luokan avulla kuten listauksessa 7.20. Toistamista varten toteutetaan myös BufferNeeded-takaisinkutsu, jossa täytetään puskuri audiodatalla ja lähetetään se audiolaitteen toistettavaksi.

**Listaus 7.20:** Suoratoisto XNA:n audio-rajapinnan avulla

```
1       ...
2       dynamicSound = new DynamicSoundEffectInstance(sampleRate,
3         (AudioChannels)channels);
4       bufferSize = dynamicSound.GetSampleSizeInBytes(TimeSpan.FromMilliseconds(50));
5       dynamicSound.BufferNeeded +=
6         new EventHandler<EventArgs>(DynamicSound_BufferNeeded);
7
8       // Tässä käynnistettäisiin taas XNA:n päivityssilmukka (ks. aiempi esimerkki)
9
10      // Toistetaan näyte
11      dynamicSound.Play();
12    }
13
14    // Täytetään puskuri
15    void DynamicSound_BufferNeeded(object sender, EventArgs e) {
16      dynamicSound.SubmitBuffer(byteArray, position, bufferSize);
17
18      position += bufferSize;
19      if (position + bufferSize > byteArray.Length) {
20        position = 0;
21        dynamicSound.Stop();
22      }
23    }
24  }
```

## 7.2 Virhearviot

Mittauksille ominaista on, että latenssit vaihtelevat prosessorin kuormasta riippuen. Koska mitään absoluuttista fysikaalista "latenssivakiota" ei ole olemassa, on virheenkin määrittäminen hankalaa. Vaikka ylimääräiset taustaprosessit pyrittiin pitämään minimissä sulkemalla tarpeettomat sovellukset, ei prosessorikuorman minimoimiseen kiinnitetty erityistä tarkkaavaisuutta. Yhdessä keskihajonnan kanssa keskiarvo antaa kuitenkin riittävän tarkkuuden, jotta voidaan todeta latenssin suuruusluokka ja alustan soveltuvuus latenssikriittisiin sovelluksiin. Keskihajonta on arvokas tunnusluku lisäksi siitä syystä, että se paljastaa isot latenssinvaihtelut. Vaihtelut saattavat olla ikäviä sovelluksissa, joissa soitannollisen rytmin ylläpitäminen on tärkeää. Toisaalta eri formaattien kohdalla keskihajonnassa tuntui olevan paljon vaihtelua, joten koko otoskannan keskihajonta onkin parempi mitta tässä yhteydessä kuin minimihajonta.

Karkeaa virhettä syntyi toisinaan lähinnä kirjausvirheen tuloksena. Esimerkiksi näppäiltäessä lukemaa mittauspöytäkirjaan jonkin numeron jääminen tuloksesta tai numeroiden järjestyksen vaihtuminen saattoi aiheuttaa hyvinkin ison virheen. Tosin tällaiset tilanteet oli yleensä helppo huomata poikkeavina keskiarvoina tai hyvin suurena hajontana, jolloin mittaustulos voitiin helposti hylätä. Jos näppäilyvirhe on ollut kohtalaisen lähellä keskiarvoa, on virheellisiä arvoja voinut jäädä tuloksiin.

Jonkin verran virhettä syntyi myös manuaalisesta latenssin määrittämisestä Audacitylla. Kaikki määritykset pyrittiin tekemään käyttäen samaa suurennustasoa, mutta se ei kuitenkaan suurten latenssivaihtelujen vuoksi ollut mielekästä. Esimerkiksi kun 3000 näytettä sisältävä latenssi suurennetaan mahdollisimman lähelle siten, että se vielä kokonaisuudessaan näkyy ruudulla, vastaa yhden pikselin siirtymä valinnassa 4 näytettä. Nauhoitteen näytteenottotaajuuden ollessa 44100 Hz on virhe siis noin 0,09 millisekuntia eli häviävän pieni. Virhe toki kasvaa suoraan verrannollisesti latenssin kasvaessa, sillä myös näytöllä näkyvissä oleva ajanjakso pitenee. 10000 näytteen jaksolla pikselin siirtymä vastaa 16 näytettä, mutta on edelleen vain reilun millisekunnin kolmanneksen.

Hieman merkittävämpi virhe syntyy valittaessa näytön napautuksesta aiheutuvan piikin ja audionäytteen alkamisajankohtaa. Napautuksen aaltomuodosta valittiin ensimmäinen huomattavan korkea piikki nauhoitteen aaltomuodossa. “Huomattavan korkea” on kuitenkin jokseenkin epämääräinen käsite ja joskus kohdan valinta saattoikin olla jossain määrin epäkonsistenttia. Piikkien väli on luokkaa 20–50 näytettä, joten edelleen puhutaan maksimissaan vain reilun millisekunnin poikkeamasta.

Viivettä syntyy myös äänen kulkiessa ilmassa ennen kuin se saavuttaa mikrofonin. Tällä ei kuitenkaan käytännössä ole juurikaan vaikutusta latenssin määrityksessä, sillä viive vaikuttaa koko näytteeseen. Äänipiikkien välinen erotus ei muutu, vaikka mikrofonin etäisyys laitteeseen vaihtelisi mittausten välillä. Käyttäjälle etäisyyden aiheuttamalla viivelläkin voi olla hieman merkitystä, sillä käyttäjälle hermoratoja pitkin kulkeva kosketustuntemus on käytännössä välitön, mutta äänen kantautuminen korvaan vie hekten aikaa. Toki kaikki on suhteellista ja kokonaislatenssiin verrattuna äänen kantautumisen viive on vielä mitättömän pieni.

Hieman virhettä syntyy toki mikrofonin sijoittelusta äänilähteisiin nähden. Mittauksissa on käytännössä läsnä kaksi äänilähdettä: laitteen kaiutin sekä näytön napautuskohta. Näiden etäisyydet mikrofonista eroavat arviolta 3–7 cm. Ilmassa viivettä syntyy 34 cm matkalla noin 1 ms, joten virhe vaihtelee välillä 0,1–0,2 ms

Jos edellä löydetyt laskettavissa olevat virhelähteet otetaan huomioon on yksittäisen mittauksen maksimivirhe noin 2 millisekuntia.

## 7.3 Arviointi

Mittaukset paljastivat huomattavia eroja alustojen välillä. Jos tarkastellaan pelkästään laitteiden suoritintehoja tai hintalappua, ei audion toistollisesta suorituskyvystä voida vetää suoria johtopäätöksiä. Toki tehokkaasta suorittimesta on etua etenkin syntetisoitaessa ääntä reaaliaikaisesti, mutta tavanomaiset audion toistoon liittyvät toimenpiteet on mahdollista suorittaa myös heikkotehoisemmilla laitteilla ilman suurta latenssia. Alustalla ja audiolaitteistolla näyttäisi olevan huomattava vaikutus. Näihin ohjelmistokehittäjä ei tosin oikein voi vaikuttaa. Sen sijaan käyttötarkoitukseen sopivan kirjaston valinnalla voi latenssia pienentää tuntuvasti, jos se on tarpeen.

Lähes kaikille alustoille on tarjolla helppokäyttöinen, mutta rajoittunut rajapinta, sekä suoratoiston mahdollistava monipuolisempi rajapinta. Suoratoistorajapinta tarjoaa monesti mahdollisuuden muokata muun muassa puskureiden kokoa ja ulostulevan audion näytteenottotaajuutta alhaisen latenssin toivossa. Pelkkä latenssi ei kuitenkaan kerro koko totuutta. Vaikka tietyllä kirjastolla olisi mahdollista saavuttaa huomattavasti muita vaihtoehtoja alhaisempi latenssi, ei aina esimerkiksi usean äänileikkeen toistaminen yhtäaikaisesti olekaan mahdollista. Toisaalta kirjaston vaihtaminen ei aina ole järkevää, vaikka latenssi hieman pienenisi, sillä uuden kirjaston käyttö voi olla merkittävästi hankalampaa saavutettavaan etuun nähden.

Näin yleensä onkin suoratoistoa tukevien kirjastojen kohdalla. Kehittäjä voi esimerkiksi ottaa käyttöön liian pienen audiopuskurin. Vaikka äänentoisto vaikuttaisi ensiksi hyvältä, saattaa suoritinkuorman kasvaessa tai laitteiston vaihtuessa ilmetä herkästi äänen pätkimistä. Niinpä kehittäjän voikin olla työlästä löytää kaikilla tuetuilla laitteilla toimiva konfiguraatio tai toteuttaa adaptoituva mekanismi, joka säätelee puskurin kokoa tarpeen mukaan. Lisäksi äänileikkeiden lataaminen aiheuttaa toisinaan jonkin verran ylimääräistä vaivaa, jos tarjolla ei ole valmiita työkaluja esimerkiksi pakatun audion purkamiseen tai näytteen formaatin selvittämiseen. Usein onkin huomattavasti helpompaa tyytyä vain käyttämään korkean tason kirjastoa.

Voitaneen todeta, että tutkimuksessa päästiin tavoitteeseen ja tuloksia voidaankin heti myös soveltaa käytännössä Drumkit-esimerkkisovellusten yhteydessä. Oletettavaa kuitenkin on, että tulokset vanhenevat nopeasti, sillä älypuhelinvalmistaja tuovat jatkuvasti uusia laitemalleja markkinoille. Tästä hyvänä esimerkkinä on Lumia 800, joka otettiin mukaan vasta ihan mittausten loppumetreillä. Muutenkaan käytetty laitekanta ei ollut lopulta kovinkaan kattava. Kuten tuloksista nähdään, laitteiden välillä olevat erot voivat olla isojakin. Niinpä tuloksia ei välttämättä voida täysin yleistää koskemaan kaikkia tiettyä alustaa käyttäviä laitteita. Esimerkiksi ZTE Bladella paras tulos saavutettiin OpenSL-kirjastoa käyttäen, kun taas Samsungin Nexus S:llä se tuotti puolestaan heikoimman tuloksen.

# 8. YHTEENVETO

Tämän diplomityön tavoitteena oli selvittää mille mobiilialustoille on tällä hetkellä mahdollista toteuttaa alhaista audiolatenssia vaativa sovellus, ja mitkä seikat yleisesti vaikuttavat audiolatenssiin. Samalla ajatuksena oli kartoittaa ja arvioida eri alustoille tarjolla olevia audiokirjastoja ja toteuttaa niiden avulla pieniä testisovelluksia latenssimittauksia varten.

Mitattava suure oli kokonaislatenssi, joka koostuu kosketustapahtuman viiveestä, audiotoiston viiveestä, sekä muun suoritinkuorman tuomasta viiveestä. Latenssien perusteella voitiin laskea otoksille myös keskihajonta, joka kertoo latenssien keksimääräisen poikkeaman keskiarvosta. Tämä on arvokas tieto Drumkitin kaltaisissa sovelluksissa, joissa soitannollisen rytmin ylläpitäminen on keskeistä. Mittausmenetelmä havaittiin hyväksi ja tarpeeksi yksinkertaiseksi suuren määrän toistoja vaativaa tutkimusta varten. Arvioidun 3000 tuhannen mittauksen sijaan onnistuneita latenssimäärityksiä kertyi yhteensä 2684 kappaletta.

Mittausten pohjalta voitiin todeta, että testiryhmän alustoista parhaan audiolatenssin omaavat MeeGo 1.2 Harmattan, Series 40, iOS ja Windows Phone 7. Sen sijaan älypuhelin markkinoita dominoiva Android, sekä kolmanneksi myydyin älypuhelinalusta Symbian, pärjäsivät heikoimmin. Käytännössä kummallekaan alustalle ei ole mahdollista tuottaa latenssikriittisiä audiosovelluksia. BlackBerry 6.0 sijoittuu näiden kahden ryhmän väliin, mutta senkään tulos ei ole erityisen hyvä. Myös laitteiden välillä on luonnollisesti eroja, eikä esimerkiksi laitteen suoritintehon tai hinnan perusteella voida tehdä suoria johtopäätöksiä audion toiston toimivuudesta.

Teoreettisesta näkökulmasta audiopuskurin koko sekä näytteenottotajuus vaikuttavat suuresti aikaan, joka kuluu ennen kuin digitaalinen audionäyte päätyy ohjelmasta audiolaitteiston toistettavaksi. Toisinaan kuitenkin alusta rajoittaa audion toistoa siinä määrin, ettei edes puskurikoon minimoimisesta huolimatta ole mahdollista saavuttaa kelvollista latenssia. Monesti tilanne on myös se, että puskurin kokoon vaikuttamiseksi on käytettävä suoratoistorajapintaa, jonka käyttöönotto on pääsääntöisesti huomattavan hankalaa verrattuna korkeamman tason rajapintoihin.

Tehtyjen havaintojen nojalla, voitaneen todeta, että tutkimus tavoitti sille asetetut tavoitteet. Oletettavaa kuitenkin on, että mittaustulosten merkitys pienenee nopeasti, sillä älypuhelinvalmistaja tuovat jatkuvasti uusia laitemallejaan markkinoille. Audiontoistolliset periaatteet kuitenkaan tuskin kokevat suuria muutoksia ihan piak-

koin ja näin ollen tehtyjä päätelmiä voitaneen soveltaa vielä pitkälle tulevaisuuteen. Käytetty laitekanta ei ollut lopulta kovinkaan kattava, mutta sen laajentaminen onnistuisi helposti etenkin testisovellusten toteutustyö on jo tehty. Nähtäväksi myös jää kuinka pian markkinoille ilmestyy kokonaan uusi alusta ja miten esimerkikisi lisääntyvä HTML5:n käyttö mobiilisovelluksissa tulee vaikuttamaan latensseihin.

# LÄHTEET

[1] A Tasty Pixel. Loopy$^2$. [WWW]. [Viitattu 14.09.2011]. Saatavissa: http://loopyapp.com/

[2] Apple inc. Logic: Learn about I/O buffer size and monitoring latency. [WWW]. [Viitattu 23.10.2011]. Saatavissa: http://support.apple.com/kb/HT1314

[3] Apple inc. iOS Developer Library. [WWW]. [Viitattu 12.09.2011]. Saatavissa: http://developer.apple.com/library/ios/

[4] Blanchette, J. & Summerfield, M. 2006. C++ GUI programming with Qt 4. Stoughton, MA, Prentice Hall in association with Trolltech Press. 537 s.

[5] Blass, E. Windows Phone 7 Now Runs Native Code; Jailbreak Imminent? [WWW]. [Viitattu 23.09.2011]. Saatavissa: http://pocketnow.com/windows-phone/windows-phone-7-now-runs-native-code-jailbreak-imminent

[6] Chehimi, F., Stichbury, J. & Cartwright, S. 2008. Games on Symbian OS: A Handbook for Mobile Development. Symbian Press. Chichester, John Wiley & Sons. 373 s.

[7] Connor, D. Dealing with Latency: The Audio Buffer. [WWW]. [Viitattu 19.10.2011]. Saatavissa: http://thestereobus.com/2007/12/13/dealing-with-latency-the-audio-buffer/

[8] Creative Labs. OpenAL. [WWW]. [Viitattu 22.02.2012]. Saatavissa: http://connect.creativelabs.com/openal/

[9] Dolecules, M. Everyday Looper : The loop station tailor-made for the iPhone. [WWW]. [Viitattu 26.09.2011]. Saatavissa: http://www.mancingdolecules.com/everyday-looper/

[10] Evers, J. Microsoft to phase out Pocket PC, Smartphone brands. [WWW]. [Viitattu 13.09.2011]. Saatavissa: http://www.infoworld.com/d/hardware/microsoft-phase-out-pocket-pc-smartphone-brands-232

[11] Fitzek, F., Mikkonen, T. & Torp, T. 2009. Qt for Symbian. Chichester, John Wiley & Sons. 208 s.

[12] Fling, B. 2009. Mobile Design and Development: Practical Concepts and Techniques for Creating Mobile Sites and Web Apps. O'Reilly Series. Sebastopol, CA, O'Reilly first edition painos. 309 s.

[13] Gartner Inc. Gartner Says Worldwide Smartphone Sales Soared in Fourth Quarter of 2011 With 47 Percent Growth. [WWW]. [Viitattu 05.03.2012]. Saatavissa: http://www.gartner.com/it/page.jsp?id=1924314

[14] Google Inc. Android developers portal. [WWW]. [Viitattu 12.09.2011]. Saatavissa: http://developer.android.com

[15] Ivan. Android Audio: Problems, Hidden Limitations and OpenSL ES. [WWW]. [Viitattu 02.02.2012]. Saatavissa: http://mindtherobot.com/blog/555/android-audio-problems-hidden-limitations-and-opensl-es/

[16] Keltz, A. Opening Pandora's Box? – The "L"word - latency and digital audio systems. [WWW]. [Viitattu 23.10.2011]. Saatavissa: http://whirlwindusa.com/support/tech-articles/opening-pandoras-box/

[17] Kerris, N. & Dowling, S. Apple Reinvents the Phone with iPhone. [WWW]. [Viitattu 08.09.2011]. Saatavissa: http://www.apple.com/pr/library/2007/01/09Apple-Reinvents-the-Phone-with-iPhone.html

[18] Lago, N. P. & Kon, F. The Quest for Low Latency. [WWW]. [Viitattu 23.10.2011]. Saatavissa: http://gsd.ime.usp.br/~lago/masters/latency-paper.pdf

[19] Lecrenski, N., Watson, K. & Fonseca-Ensor, R. 2011. Beginning Windows Phone 7 Application Development: Building Windows Phone Applications Using Silverlight and XNA. Wrox Programmer to Programmer. Indianapolis, IN, John Wiley & Sons. 576 s.

[20] Lee, H. & Chuvyrov, E. 2011. Beginning Windows Phone 7 Development. Apress Series. Apress L. P. 512 s.

[21] MasterDroid. Intro to the three Android Audio APIs. [WWW]. [Viitattu 25.02.2012]. Saatavissa: http://www.wiseandroid.com/post/2010/07/13/Intro-to-the-three-Android-Audio-APIs.aspx

[22] Meier, R. 2010. Professional Android 2 Application Development. Indianapolis, IN, John Wiley & Sons. 576 s.

[23] Newsroom Intel. Intel and Nokia Merge Software Platforms for Future Computing Devices. [WWW]. [Viitattu 07.09.2011]. Saatavissa: http://www.intel.com/pressroom/archive/releases/2010/20100215corp.htm

[24] Nokia Corporation. Nokia Developer. [WWW]. [Viitattu 21.09.2011]. Saatavissa: https://www.developer.nokia.com

[25] Nokia Corporation. Series 40 Platform. [WWW]. [Viitattu 07.09.2011]. Saatavissa: https://www.developer.nokia.com/Devices/Series_40/

[26] Nokia Corporation and/or its subsidiaries. Qt. [WWW]. [Viitattu 15.09.2011]. Saatavissa: http://qt.nokia.com/

[27] Nokia Corporation and/or its subsidiaries. Qt GameEnabler. [WWW]. [Viitattu 16.09.2011]. Saatavissa: https://projects.developer.nokia.com/qtgameenabler

[28] Open Handeset Alliance. Industry Leaders Announce Open Platform for Mobile Devices. [WWW]. [Viitattu 12.09.2011]. Saatavissa: http://www.openhandsetalliance.com/press_110507.html

[29] Press Services Nokia. Nokia to acquire Trolltech to accelerate software strategy. [WWW]. [Viitattu 07.09.2011]. Saatavissa: http://qt.nokia.com/about/news/archive/press.2008-01-28.4605718236/

[30] Press Services Nokia. Nokia reaffirms commitment to Symbian platform. [WWW]. [Viitattu 07.09.2011]. Saatavissa: http://press.nokia.com/2010/11/08/nokia-reaffirms-commitment-to-symbian-platform-2/

[31] Press Services Nokia. Nokia and Accenture finalize Symbian software development and support services outsourcing agreement. [WWW]. [Viitattu 07.09.2011]. Saatavissa: http://press.nokia.com/2011/06/22/nokia-and-accenture-finalize-symbian-software-development-and-support

[32] PulseAudio Team. PulseAudio. [WWW]. [Viitattu 16.09.2011]. Saatavissa: http://www.pulseaudio.org/

[33] R., L. Announcing BBX - The Next Generation BlackBerry Platform. [WWW]. [Viitattu 02.02.2012]. Saatavissa: http://blogs.blackberry.com/2011/10/bbx-blackberry/

[34] Research In Motion. BlackBerry Developer Zone. [WWW]. [Viitattu 02.02.2012]. Saatavissa: http://us.blackberry.com/developers/

[35] Savolainen, H. More moving parts is better???? [WWW]. [Viitattu 19.10.2011]. Saatavissa: http://4front-tech.com/hannublog/?page_id=24

[36] Smith, S. 1997. The scientist and engineer's guide to digital signal processing. San Diego, CA, California Technical Publishing. 626 s.

[37] Smule Inc. Smule: Experience Social Music. [WWW]. [Viitattu 26.09.2011]. Saatavissa: http://www.smule.com

[38] Sood, A. Introducing Nexus One. [WWW]. [Viitattu 13.09.2011]. Saatavissa: http://googlemobile.blogspot.com/2010/01/introducing-nexus-one.html

[39] Sun Microsystems. The Basics of J2ME. [WWW]. [Viitattu 20.09.2011]. Saatavissa: http://developers.sun.com/mobility/midp/chapters/muchowcore/ch1.pdf

[40] Tseng, E. The first Android-powered phone. [WWW]. [Viitattu 13.09.2011]. Saatavissa: http://googleblog.blogspot.com/2008/09/first-android-powered-phone.html

[41] Tyson, M. Using RemoteIO audio unit. [WWW]. [Viitattu 06.09.2011]. Saatavissa: http://atastypixel.com/blog/using-remoteio-audio-unit/

[42] Tyson, M. Developing Loopy, Part 2: Implementation. [WWW]. [Viitattu 26.09.2011]. Saatavissa: http://atastypixel.com/blog/developing-loopy-part-2-implementation/

[43] Wang, G. Designing Smule's iPhone Ocarina. [WWW]. [Viitattu 26.09.2011]. Saatavissa: https://ccrma.stanford.edu/~ge/publish/ocarina-nime2009.pdf